2020

Enhancing Video Streaming Quality of DASH over Cloud/Edge Integrated Networks

Mohammedameen, Ibrahim

Enhancing Video Streaming Quality of DASH over Cloud/Edge Integrated Networks

By

Ibrahim S. Mohammedameen

A thesis submitted to the University of Plymouth

In partial fulfilment for the degree of

Research Masters

School of Engineering, Computing and Mathematics

April 2019

April 2019

Declaration

At no time during the registration for the degree of *Research Masters* has the author been registered for any other University award without prior agreement of the Doctoral College Quality Sub-Committee. Work submitted for this research degree at the University of Plymouth has not formed part of any other degree either at the University of Plymouth or at another establishment.

This is to certify that the candidate, Ibrahim Sh. Mohammedameen carried out the work submitted herewith

Candidate's Signature:

Ibrahim Sh. Mohammedameen _____ Date: 23/04/2019

Supervisor's Signature:

Dr. Lingfen Sun _____ Date: 23/04/2019

Number of words: 13897

Copyright & Legal Notice

Acknowledgment

I wish to extend my warmest thanks and appreciation to those who have helped me during my thesis work.

Many thanks to Dr. Lingfen Sun, for her continues support and guidance during my research work, it was not possible without her guidance and inspiration from the start to the end of this Research degree.

Dr. Is-Haka Mkwawa thank you very much for your help in every step, following and guiding me during every step of my test, and testbed setup.

My family, for their continues support during my journey, continues love and always believing in me, my father for making this a living reality for me instead of a dream, without them none of this would have been even possible.

Publications

Presented a paper in (*SCI 2019, Aug)* conference under 'Follow-me Prefetching for Video Streaming over Mobile Edge Computing Networks' title. Attached to the thesis as appendix 1.

Abstract

With the advancement of mobile technologies and the popularity of mobile devices, mobile video streaming applications/services have increased considerably in recent years. Dynamic Adaptive Streaming over HTTP (DASH) or MPEG-DASH is one of the most widely used video streaming techniques over the Internet. It adapts video sending bit rate according to available network resources, however, in case of low bandwidth, DASH performs poorly, which will cause video quality degradation and video stalling.

Mobile Edge Computing (MEC) or Multi-access Edge Computing, in connection with the backend cloud has been used to reduce latency and overcome some of the video quality degradation problems for mobile video streaming services. However, an end user might be suffering from video quality drop downs when s/he moves out from the coverage of one node to another or when the mobile network condition goes down. To tackle the degradation problems and assure enhanced video streaming quality, a novel follow-me Edge Node Prefetching (ENP) scheme was proposed and developed in the project, by prefetching video segments in advance in the upcoming node used by the end-user. A test bed was set up consisting of a backend cloud (OpenStack), two edge nodes (LXD Containers) and one mobile device, the ENP algorithm was implemented on the cloud server and client sides. Experiments were carried out for the DASH streaming service based on Dash.js from the DASH Industry Forum. Preliminary results show that the ENP scheme can maintain higher video quality and less service migration time when moving from one mobile node to another, when compared to existing approaches, i.e. live migration in Follow-me-Edge and the C-up schemes. The proposed scheme could be useful in smart city applications or providing seamless mobile video streaming services in Cloud/Edge integrated networks.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

M2M	Machine-to-Machine
MEC	Mobile Edge Computing
IOT	Internet of Things
QoS	Quality of Service
QoE	Quality of Experience
OTT	Over-The-Top
HLS	HTTP Live Streaming
MOS	Mean Opinion Score
D2D	Device-to-Device
HEVC	High efficient Video Coding
NAT	Network Address Translation
RTP	Real Time Streaming Protocol
UDP	User Datagram Protocol
HTTP	Hyper Text Transfer Protocol
MPD	Media Presentation Description
MF	Manifest File
DF	Data Frame

1. Introduction

1.1 Introduction

With the advancement of mobile communications and video processing technologies in recent years, video streaming services (such as those provided by YouTube and Netflix) have become one of the most dominant services over the Internet. According to the latest Cisco Visual Networking Index (VNI), video services will account for about 82% of all consumer Internet traffic by 2021, growing threefold between 2016 to 2021 [1]; mobile video will be about 78% of the world's mobile data traffic by 2021, increasing 9-fold between 2016 and 2021 [1]. In this context, great efforts have been madeto improve the quality of video streaming services over the mobile Internet.

Over the years, Over-The-Top (OTT) service providers such as YouTube and Netflix have been using the MPEG- DASH (Dynamic Adaptive Streaming over HTTP) standard to deliver video streaming services over the Internet [2]. MPEG-DASH has many advantages such as flexible quality adaptation and simplicity in implementation due to its easy integration with the existing HTTP-based infrastructure. In MPEG-DASH, the video is prepacked with short video segments for different representations corresponding to different video qualities at the server. A DASH client predicts available network bandwidth and sends an HTTP-request to the server for appropriate video representation segments. Due to varying network conditions, an ideal DASH scheme should be able to adapt between different video segments to provide acceptable video quality and hence, acceptable Quality of Experience (QoE) to video streaming customers. Normally video contents are located in servers in Data Centers over the Cloud and/or in cache servers provided by Content Delivery Networks (CDN). When a client is too far away from the server (e.g. across a continent), latency and network congestion will have adverse effects on video streaming quality. The situation will get worse when the Internet video is streamed to mobile devices.

Mobile Edge Computing (MEC), a promising technology for 5G mobile networks, has been utilised to assist video streaming services due to its ability to facilitate the provisioning of high data rate and low latency services to end users and its ability to provide computational power at the mobile edge. It has been used for MPEG-DASH to reduce the backbone traffic to the Cloud and bring video contents close to the end user. The MEC specified by ETSI [3], is one of the Mobile Edge Network (MEN) structures that offer intense computing capability along with real-time communication ability with the end user.

There are many elements that make on-demand video delivery possible. One of them is the Bandwidth. The best way to understand bandwidth is to describe the bandwidths as highways, the video information as cars, and the internet as highways, so the car can arrive faster and easier if no other cars or if the highway is wider. The internet takes the same concept to work, if you are

the only person downloading a video or a file it will transfer the data faster than if more than one person trying to get the same video or file.

The bandwidth concept is same, imagine the bandwidth as the number of lanes on the highway, if the bandwidth is small, the traffic will be slow, and vice versa if the bandwidth of the website is large, information can travel both ways smoother and faster, and for video delivery purposes bandwidth is very important because to send a high quality video and audio over the internet it requires a high amount of bandwidth.

After the bandwidth comes the streaming audio and video, streaming helps the user to watch on-demand videos without the need of downloading them to their personal devices or computers.

Streaming relies on the following options: -

1. Server to host the video.
2. The user requesting the video and watching it.
3. The server responding to the users request by sending all of the video segments or pieces accordingly using streaming media protocols.
4. Finally, a player on your device, to decode and play the video for you, Firefox, VLC are two known ones.

On demand video delivery is promising quite a lot, but it does not come without limitations, like if the user has slow internet the data won't arrive to the user fast enough, or it will cause video quality degradation because of the slow internet, and also some websites are not able to provide enough bandwidth to their users. In recent years, many services including video streaming services have moved to the cloud, which may create new problems and new challenges as datacentres might be far away to the end users, and latency and core network bandwidth consumption might be an issue.

Mobile Edge Computing (MEC) provides a promising approach to move some of the computing facilities close to the mobile end user, thus, reducing the latency when delivering services to the end user and also reducing the load to core network bandwidth consumption as in the case of cloud services. How to combine MEC with backend cloud services to enhance the quality of mobile video streaming remains an open and challenging question.

1.2 Aim and Objectives

This project aims to design and develop an approach for mobile video streaming services by combining the strength of both mobile edge computing and backend cloud services.

The project consists of the following four objectives.

1. Carry out literature review of the state-of-the-art
2. Setup a testbed including MEC and backend cloud for DASH video streaming based on open-source tools/software
3. Carry out experiments to assess the quality of DASH video streaming based on existing approaches
4. Propose new solutions/approaches for video streaming services and compare the performance with those of the existing ones.

1.3 Structure of the thesis

This thesis is divided into 3 major parts:

Chapter 2 provides an outline of the current literature in delivering Video quality services, cloud serves how to combine both of them to ensure QoE to the end-user.

Chapter 3 devoted to the approaches and the research methodology used by the author for carrying out the experiments in the research, which provides the approach algorithm, details step by step installation and each part configuration.

Chapter 4 presents the results of the tests that have been carried up for approaches, discussion and analysing the results, finally the conclusion and future work in chapter 5 shows the limitation of the work and how to improve it for future work, with a conclusion of the thesis.

2.Literature Review

2.1 Background

The delivery of mobile content to users is the main motive of network systems. At present, the delivery of high definition videos with high resolution has become the prime focus to enhance the context of 5G network development in the future. The 5G network enables increased network capacity along with the QoE by adding required network intelligence in all type of network requirements.

2.1.1 Parameters affecting video quality

Delivering high-quality videos to the end user over HTTP networks is still a big challenge to the service providers due to the number of the parameters affecting video quality, those parameters are: -

Coding parameters: - encoding parameters controls the amount of quality losses during encoding stages, which depends on the encoding algorithms e.g. (MPEG, H26x), Frame rate, Bitrate.

Network parameters: - those parameters affect the video during streaming or transmissions e.g. (delay, and delay variation (Jitter)).

Other parameters: - other parameters such as image size, colour, motion might have huge effect on the video quality.

Frames per second (FPS), resolution, and clarity also have effects on video quality your screen resolution and how many frames per second you receive, but our prime focus in this study is on the network parameters that affects video quality.

2.1.2 Quality of Experience methods

Measuring the delight or the annoyances of the video being watch by the end user is what we call Quality of Experience, and the most widely used metric to measure the QoE level for the end user is the mean opinion score (MOS).

2.1.2.1 Subjective methods

Subjective methods are rated using the ITU-T (International Telecommunication Standardization Sector) scale which is called MOS (Mean Opinion Score), table -1- shows the range of MOS score from 5 to 1.

MOS	Quality	Impairment
5	Excellent	High Quality
4	Good	Not annoying
3	Fair	Slightly annoying
2	Poor	Annoying
1	Bad	Very annoying

Table -1- MOS SCORE

2.1.2.2 Objective methods

Objective methods are based on mathematical algorithms rather than scales, which cloud be automated or generated using mathematical equations, the usage of this method has been increased is the last decades [21] because it give more reliable and close to reality results.

2.2 Video streaming methods

2.2.1 Real time streaming protocol (RTP)

Real Time Streaming Protocol (RTP), is one of the oldest techniques for video streaming. It is based on User Datagram Protocol (UDP), and as known RTP uses Push protocol, that means the RTP server pushes the video data to the end-user, but it has a lot of disadvantages like it could pass network

address translation (NAT) or even firewalls unless the right codec is used to support scalable video transmission e.g. Scalable Video Codec (SVC)[3].

2.2.2 Progressive downloading

This method is very close to video downloading, it is based on hypertext transfer protocol (HTTP), it has advantages that RTP does not has like it can be used cashes, CDNs, Proxies and it has no issues with NAT nor firewalls unlike RTP, the disadvantages of this approach is that, the HTTP protocol increases the transmission to the double amount of media bitrate [2].

2.2.3 Adaptive HTTP streaming (AHS)

This approach came to solve the bitrate handling issues that other protocols had, and it is one of the first solutions inside 3GPP[4]. The idea is to cut media files into number of segments and encode them into different bitrate and resolutions, then provide those segments on the Web server, and download them with standard HTTP GET request, the adaptation for the bitrate and resolution is done on the client side, the client can switch to higher resolutions per segment, if s/he has enough network resources. AHS uses Media Presentation Description (MPD) to combine the segments, segment orders and segment bitrate.

2.2.4 Dynamic Adaptive HTTP Streaming (DASH)

The beginning of the MPD file has all different bitrate for each segment, starting time, and the duration of each segment, a client would browse the MPD file first, with the information inside the MPD file it will receive the individual segments according to the available bandwidth. in case of bandwidth change the end-user can easily receive another representation that fits the available bandwidth, other companies like apple HTTP live streaming [6], adobe dynamic HTTP streaming [7], and Microsoft smooth streaming [8], offers dynamic adaptive streaming, at the beginning the end-user will request the desired MPD file start with the download of the segments and eventually adapt dynamically to bandwidth fluctuations

The system architecture is shown in Figure 1 [9] and based on that MPEG started a new work item called Dynamic Adaptive Streaming over HTTP (DASH).
On request, the manifest file will be provided to the client in order to initiate the session (cf. step-1 in Figure 9). The client will parse the manifest file and request individual segments compliant to the delivery format using HTTP and according to the information found in the manifest file (cf. step-2 in

Figure 9). For the manifest file, DASH adopted the Media Presentation Description (MPD) as defined by 3GPP AHS [3] as a starting point. The MPD follows a data model comprising a sequence of one or more consecutive non-overlapping periods for which one or more representations may be available. A single representation refers to a specific media following certain characteristics such as bitrate, framerate, resolution, etc. Furthermore, each representation consists of one or more segments that actually describe the media and/or metadata to decode and present the included media content. The delivery format defines the format of the segments to be delivered to the client upon the HTTP requests based on the MPD. Finally, as the delivery format shall be compatible to existing MPEG formats (i.e. ISOBMFF and MPEG-2 TS), it shall be also possible to provide easy conversion from and to these formats.

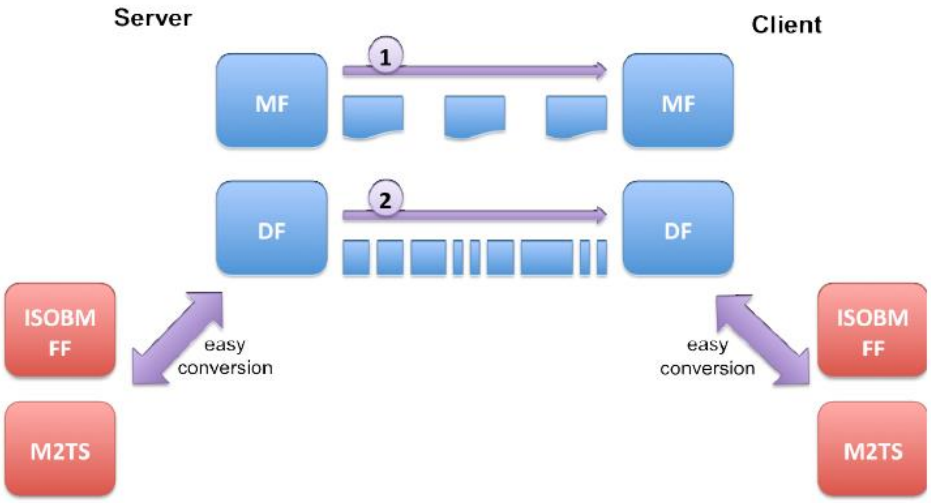


Figure -1- DASH system architecture [9]

2.3 Cloud computing

The definition of cloud computing based on NIST[NIST] is 'Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction', and the cloud computing can be deployed in different type of clouds e.g. (Public cloud, private cloud, hybrid cloud) based on the needs of the consumers.

2.3.1 Cloud computing categories

The three main cloud computing categories nowadays are: -

1- (SaaS) software as a Service: - like Microsoft 365.
2- (LaaS) Infrastructure as a Service: - like Rackspace.
3- (PaaS) Platform as a Service: - like salesforce.com

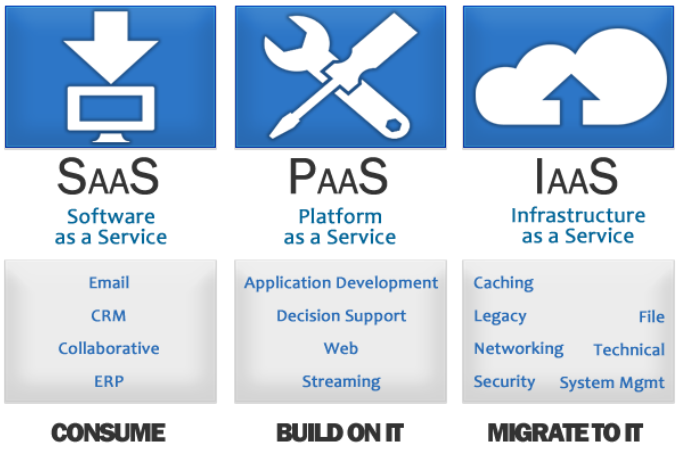


Fig -2- cloud computing categories [2]

Cloud computing system is divided into two parts: the front end and the back-end part, and they are both connected to each other through the internet.

The front end is the client side e.g. (mobile device or computer), and the back end it the cloud section.

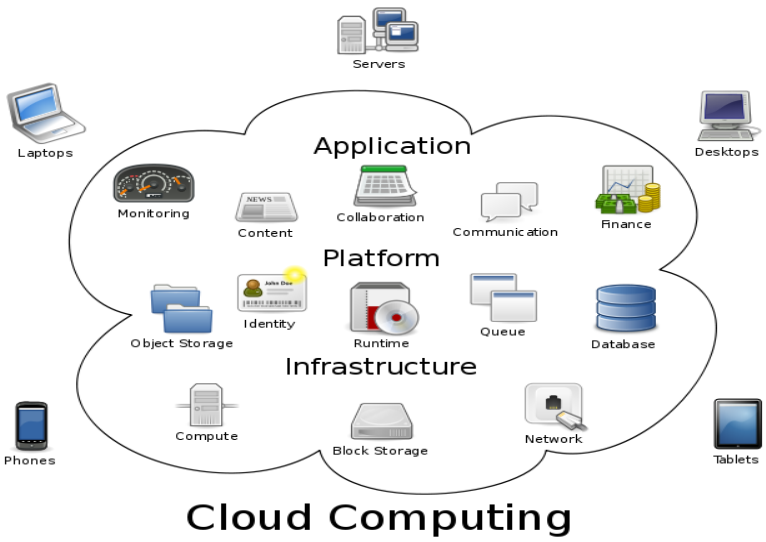


Fig-3- cloud computing infrastructure [1]

The front end consists of the client's computer or computer network. Also the application essential to access the cloud computing system. It is not necessary that all cloud computing systems have the same user interface.

On the back end of the cloud technology system, there are various computers, servers and data storage systems that make up the cloud. A cloud computing system could potentially include any computer program, from data processing to video games. Generally, each application will have its own dedicated server.

2.4 Mobile edge computing (MEC)

Multi-access edge computing (MEC) or Mobile Edge Computing, is a network infrastructure that IT services and cloud computing in the mobile edge and the Radio access network (RAN), the main advantage of MEC is preforming tasks and running applications closer to the end-user, so that the load on the network will be less and the application performs better, its implemented in the EnodeB, by that MEC ensures, less latency, better Quality of Experience, and better service delivery [MEC]. Mobile Edge Computing (MEC) [11] in particular is anticipated that intelligence and video content awareness can be implemented in the edge node for optimizing the QoE for end users.

The principal objective of MEC is to place storage and computation resources at the network edge, in the user vicinity. The data processing can be driven accordingly from inaccessible cloud to the edge. When the data is processed locally and data streams are accelerated via several techniques such as caching and compression, MEC reduces the bottleneck toward the core network. In addition, it decreases end-to-end latency, enabling the offload of important computation load from power constrained user equipment to the edge. The executive briefing of the European Telecommunications Standards Institute (ETSI) MEC initiative argues that one edge computing shall enable new computation-intensive services and shall yield promising business models. It also represents a fault resilient solution for its decentralized architecture [12].

Most of the MEC studies are focusing about reducing the latency, Taleb [15] tackles the migration problems in the context of smart cities. They proposed an approach to enhance video streaming experience for the end user in smart cities based on Follow me Edge concept, unlike our studies they are focusing on the users mobility from one node to another and not on the video quality.

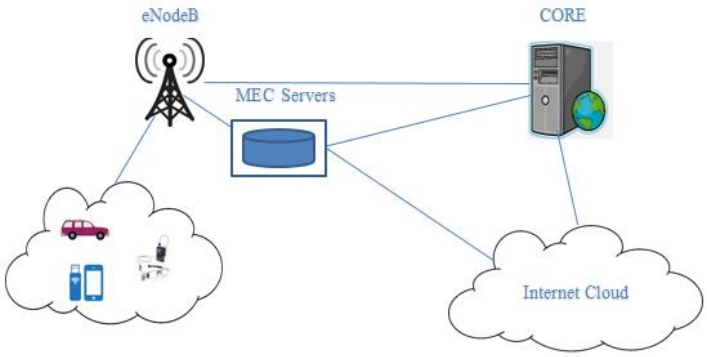


Fig-4- Mobile Edge computing concept

2.5 Ffmpeg

Ffmpeg is one of the free software's to encode videos, audios, or any other multimedia formats into different bitrate, frame rate, resolution, and sample rate, the main point of using Ffmpeg is that it almost includes all multimedia codecs to encode or decode uncommon formats to a common one[22].

2.6 System Containers

System containers are different from Docker containers, because they are configured to host a complete operating system without VMware emulators, rather than just an individual application inside the containers, there are a lot of type of system containers e.g. (openVZ containers, LXC containers, LXD containers).

2.6.1 OpenVZ containers

While virtualization technologies such as VMware and Xen provide full virtualization and can run multiple operating systems and different kernel versions, OpenVZ uses a single patched Linux kernel and therefore can run only Linux. All OpenVZ containers share the same architecture and kernel version. This can be a disadvantage in situations where guests require different kernel versions than that of the host. However, as it does not have the overhead of a true hypervisor, it is very fast and efficient.

Memory allocation with OpenVZ is soft in that memory not used in one virtual environment can be used by others or for disk caching. While old versions of OpenVZ used a common file system (where each virtual

environment is just a directory of files that is isolated using chroot), current versions of OpenVZ allow each container to have its own file system.

OpenVZ containers are still in beta version of live migration.

2.6.2 LXC containers

LXC containers are often considered as something in the middle between a chroot and a full fledged virtual machine. The goal of LXC is to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel.

LXC combines the kernel's cgroups and support for isolated namespaces to provide an isolated environment for applications. .LXC containers don’t support live migration

2.6.3 LXD containers

LXD is a next generation system container manager. It offers a user experience similar to virtual machines but using Linux containers instead. Its image based with pre-made images available for a wide number of Linux distributions and is built around a very powerful, yet pretty simple.
LXD containers fully support live migration.

3. Testbed setup

3.1 Follow me Edge Node Prefetching Scheme

The proposed system architecture for the ENP scheme is illustrated in Fig-1- The MPEG-DASH server is hosted in the cloud and the mobile edge nodes are within the proximity of the ENodeBs. The edge nodes will act as reverse Web proxies which collects data instead of the client from the requested server then send it to the client as it is from the server itself, for the user mobile clients on behalf of the MPEG-DASH server in the cloud. During streaming at the client side, if the video segments are not cached in the edge node then the edge node will request these segments from the cloud, cache them in the edge node and serve the mobile clients. If the video segments are cached in the edge node then the edge node will serve the mobile clients without requesting the segments from the cloud. In this system, the mobile clients also communicate directly with the MPEG-DASH by transmitting some computed network parameters (e.g., throughput, current client buffer size, segment number and bitrate) to the MPEG-DASH server.

When a mobile client moves e.g. from left to right as shown in Fig.1, it will cross over many different mobile edge nodes along the path. To avoid video quality drop during video streaming from one node to another, an appropriate amount of video segments will be prefetched in following-on nodes, in advance, along the path in a follow-me prefetching manner. This is to make sure that video streaming services are seamless, and the QoE for streaming services provided to the end user is well maintained during user mobility states (e.g. watching streaming video in a moving car), by following the follow-me prefetching algorithm we assure less video quality drop down and better QoE for the client.

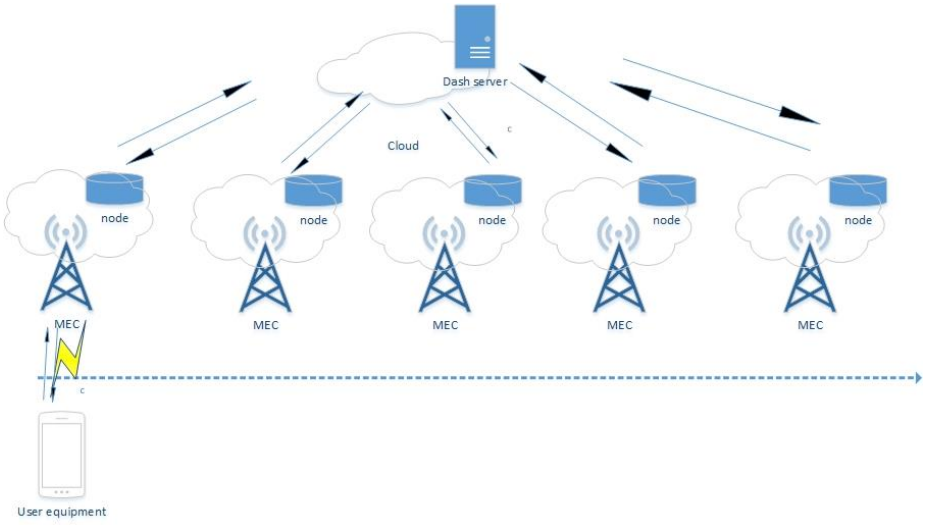


Fig-5- ENP system architecture

The proposed ENP scheme has two algorithms that run in the cloud server and mobile client side, respectively. The cloud side algorithm is listed as Algorithm 1, which starts by checking the client's buffer size, Bt (Buffer size at time t). The buffer size is sent periodically (here every 250 ms) from the client side (c.f., Algorithm 2), which fluctuates as a result of network conditions. When the current buffer size is less than the predetermined buffer size threshold (Bth), the cloud side algorithm will wake up the next closest edge node (here container C(k+1)) and start prefetching the next predetermined number of segments (ps = 5 segments in this example). After prefetching process has finished, the user will then be migrated to continue with streaming from the new edge node, the old edge node (container Ck) will then be stopped. On the client side, assuming the current serving video segment is Si and buffer size at time t is Bt, for each segment, it will calculate/estimate the current buffer size (Bt), and then send the buffer size (Bt) and the segment number (Si) to the cloud.

List of parameters used in this testbed is as described bellow:

Bitrate parameters	1080p, 720p, 460p, 360p
Buffer parameter	250ms
Number of executions of each approach	40runs per approach

Algorithm 1: The Cloud side ENP algorithm

Let t={0,...,T}
Δt= 250 ms
Current segment S_i
current container : C_k
let p_s be the number of segments to prefetch
B_t : Clients buffer size at time t
B_{th}: predefined buffer threshold
foreach Δt **do**
obtain B_t
if $B_t < B_{th}$ **then**
  wake up container $C_{(k+1)}$
  for $(j=i;\ j < i + p_s;\ j{+}{+})$ **do**
    prefetch segment S_j in C_{k+1}
  end
  stop C_k
end
;

Algorithm 2: The client side ENP algorithm

Current segment S_i
Client buffer size at time t, B_t
foreach S_i **do**
calculate B_t
send B_t to the cloud
send S_i to the cloud
;

3.2 Testbed component configurations: -

3.2.1 Containers configuration: -

LXD containers are used in this testbed as mentioned before, which are the same as virtual machines but using Linux containers, one of the main benefits of this type of containers that it supports live migration from nodes at the same host or nodes from different host, as known containers are image based and easy to pre configure before usage, that is why they are light-weighted to be moved in a timely manner period from one node to another, in this testbed we configured the containers to host Ubuntu servers with apache server installed on them to act like reverse proxy to the back-end cloud.

All containers which are in the same host has a specific amount of memory allocated to them for usage and data storage.

After installing lxd in your computer which will act like a node you will launch your first Lxd using the command bellow:

```
lxc launch ubuntu:16.04 first
```

As shown in the command you specify which operating system and which version you want to be installed in our case Ubuntu version 16.04 then you name your container, LXD containers were used in both of our presented methods, the ENP approach with live migration option and the C-up approach with instant IP change in the next containers, after installing needed you can check how many containers you have on that host by typing Sudo LXC list as shown in figure -6-:

```
lxdd@lxdd-VirtualBox:~$ sudo lxc list
+---------+---------+-----------------------+------+------------+-----------+
|  NAME   |  STATE  |         IPV4          | IPV6 |    TYPE    | SNAPSHOTS |
+---------+---------+-----------------------+------+------------+-----------+
| squid11 | RUNNING | 192.168.10.86 (eth0)  |      | PERSISTENT | 1         |
+---------+---------+-----------------------+------+------------+-----------+
| squid12 | RUNNING | 192.168.10.11 (eth0)  |      | PERSISTENT | 1         |
+---------+---------+-----------------------+------+------------+-----------+
| squid14 | RUNNING | 192.168.10.207 (eth0) |      | PERSISTENT | 1         |
+---------+---------+-----------------------+------+------------+-----------+
| squid16 | RUNNING | 192.168.10.54 (eth0)  |      | PERSISTENT | 1         |
+---------+---------+-----------------------+------+------------+-----------+
| squid2  | STOPPED |                       |      | PERSISTENT | 2         |
+---------+---------+-----------------------+------+------------+-----------+
| squid3  | STOPPED |                       |      | PERSISTENT | 1         |
+---------+---------+-----------------------+------+------------+-----------+
| squid5  | RUNNING | 192.168.10.156 (eth0) |      | PERSISTENT | 1         |
+---------+---------+-----------------------+------+------------+-----------+
| squid6  | RUNNING | 192.168.10.80 (eth0)  |      | PERSISTENT | 1         |
+---------+---------+-----------------------+------+------------+-----------+
| squid8  | RUNNING | 192.168.10.222 (eth0) |      | PERSISTENT | 2         |
+---------+---------+-----------------------+------+------------+-----------+
```

Fig-6- LXD containers List

It shows you your container names the IP address they are running on and how many snapshots you took from each container, the snapshots are needed when you use the live migration option to save the stats of all running process and resume them in the next targeted host which is done by using the checkpoint/restro properties.

After launching the needed instances we need to access using LXC exec Bash then your container name the one that you used when you created your instance first as shown in the figure -7- below:

```
lxdd@lxdd-VirtualBox:~$ sudo lxc exec squid8 bash
root@squid8:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:16:3e:43:5a:dc
          inet addr:192.168.10.222  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::216:3eff:fe43:5adc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:263 errors:0 dropped:0 overruns:0 frame:0
          TX packets:118 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:26550 (26.5 KB)  TX bytes:13958 (13.9 KB)
```

Fig-7- LXD container login

The last step to make the containers ready was to install the Reverse proxy inside the containers, we used apache and configure it to act like a reverse proxy for our backend cloud, and for the caching service in the containers we installed and configured squid to cache video segments from the cloud to the container and store them for future use, squid configuration file was edited to meet caching requirements as shown in the figure -8-, and the whole squid configuration will be in the appendix 2.

```
http_port 80 accel defaultsite=172.24.4.10 no-vhost

cache_peer 172.24.4.10 parent 80 0 no-query originserver name=myAccel


visible_hostname squid.server.comm
acl our_sites dstdomain 172.24.4.10
acl cacheDomain dstdomain 172.24.4.10
cache allow all
http_access allow our_sites
cache_peer_access myAccel allow our_sites
#Cache_peer_access myAccel deny all
acl filecachetype urlpath_regex \.m4s
acl filecachetype urlpath_regex \.html
acl filecachetype urlpath_regex \.php
acl filecachetype urlpath_regex \.txt
acl filecachetype urlpath_regex \.mpd
acl filecachetype urlpath_regex \.xml
refresh_pattern ^ftp: 1440 20% 10080

refresh_pattern ^gopher: 1440 0% 1440

refresh_pattern -i \.(gif|png|jpg|jpeg|ico)$ 10080 90% 43200 override-expire ignore-no-cache ignore-no-store ignore-private

refresh_pattern -i \.(iso|avi|wav|mp3|mp4|mpeg|swf|flv|x-flv)$ 43200 90% 432000 override-expire ignore-no-cache ignore-no-store ignore-private
refresh_pattern -i \.(m4s|mpd|xml)$ 10080 90% 43200 override-expire ignore-no-cache ignore-no-store ignore-private
refresh_pattern -i \.(txt)$ 10080 90% 43200 override-expire ignore-no-cache ignore-no-store ignore-private
```

Fig-8- squid caching configuration

3.2.2 Openstack configurations: -

As explained in the literature Openstack is one of the software's which can be used to act like cloud environment, openstack needs a lot of resources to be installed and work perfectly on a system, as you need one host for the networking, and other host for your controller and so on, and because we had some lack in resources we used Devstack (openstack) which is same as openstack but installing all components on a single host e.g.(networking, computing, storage and controller) in the same host.

After launching the devstack (openstack) successfully, you need to download the operating system image that you need for your testbed in this testbed we downloaded the minimal Ubuntu cloud image version 16.04, and the networking part of devstack which is neutron has been configured to be able to connect to the internet so the instance will be reachable in all of your network with a floating ip range of 172.24.4.0/24 with the gateway of 172.24.4.1, , is also needed to configure the rules in security group which you want your instance to use for example HTTPS, HTTP, etc.

You can launch an instance using the Devstack GUI or command line in Ubuntu, figure -9- shows the Devstack (openstack) GUI in the browser, the command to launch devstack (openstack) instances in the command line is :

```
[user@localhost]$ openstack server create –flavor flavorname –image imagename –key-name keyname.pem –security-group groupname instancename
```

Openstack by default offers you a lot of flavors to start with which is compute, storage, and memory that each instance can use, for example, we have m1.tini,m1.small and so on, m1.tini uses 1 virtual CPU, 1 GB disk, and 512 MB ram, M1.small uses 1 virtual CPU, 20 GB disk, and 2048 MB of RAM, as a server administrator is possible to create your own flavor with customized size of memory Virtual CPUs and RAM, because openstack instances does not have username and password to access it you need to use the key pair that you created every time you try to log in to the instance.

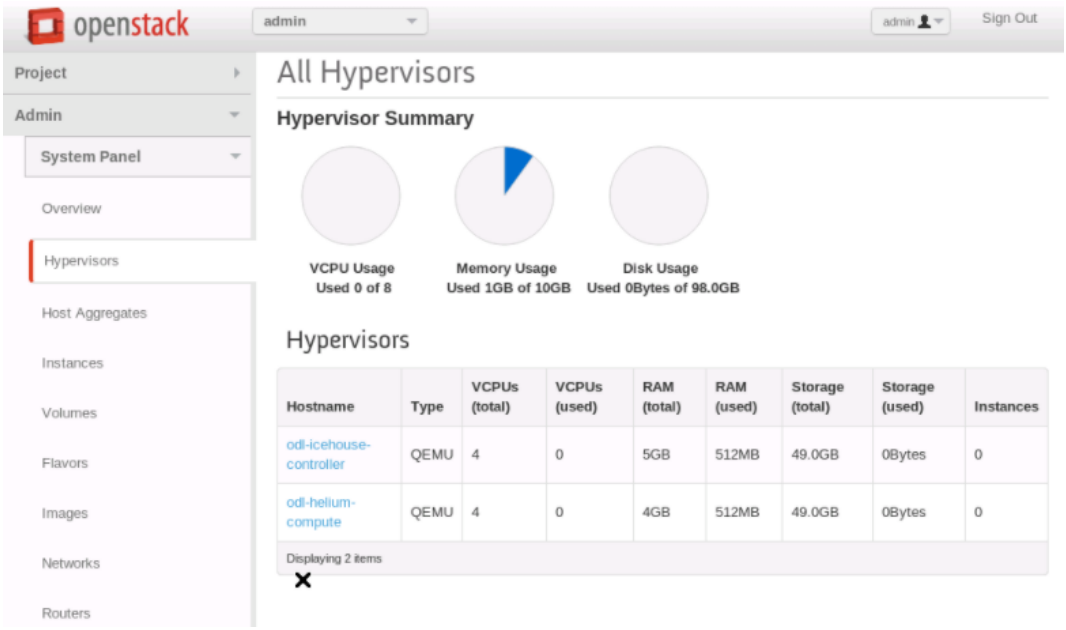


Fig-9- Devstack GUI

3.2.3 Devstack instance configuration:-

For our testbed devstack (openstack) instance is hosting the DASH server to stream the video to the end user, after installing apache server in the instance, we configure the DASH server move all created video segments to the right directory, then DASH has been configured to collect some data from the client such as: screen resolution, bitrate, throughput, client's IP, buffer size, etc, by creating a function inside DASH file, the function is shown in the figure -10-, the whole index file will be attached as appendix 3.

```
function log() {

var type = "video";

var metrics = player.getMetricsFor(type);
    var dashMetrics = player.getDashMetrics();

var streamInfo = player.getActiveStream().getStreamInfo()
var periodIdx = streamInfo.index;
    var repSwitch = dashMetrics.getCurrentRepresentationSwitch(metrics);
    var bufferLevel = dashMetrics.getCurrentBufferLevel(metrics);
    var maxIndex = dashMetrics.getMaxIndexForBufferType(type, periodIdx);
    var index = player.getQualityFor(type);
    var  throughPut = player.getThroughPut(type);
    var bitrate = repSwitch ? Math.round(dashMetrics.getBandwidthForRepresentation(repSwitch.to, periodIdx) / 1000) : NaN;
    var droppedFPS = dashMetrics.getCurrentDroppedFrames(metrics) ? dashMetrics.getCurrentDroppedFrames(metrics).droppedFrames : 0;


var output = type+" "+bufferLevel +" "+bitrate+" "+throughPut+" "+navigator.appCodeName+" "+info.browserName()+" "+info.sizeScreenW()+" "+info.sizeScreenH();

writeStats(output);

}
```

Fig-10- DASH index

After collecting the data another script has been created to send the collected data from dash and write it in a separate text file for further usage, the scripts works as a loop it prints time and print the collected data from the DASH function, script is shown in figure -11-.

```php
<?php
$d=date("d-m-Y H:i:s");
$txt = $d." ".$_GET['stats']."\n";

$myfile = fopen("stats.txt", "a") or die("Unable to open file!");
fwrite($myfile, $txt);
fclose($myfile);
?>

```

Fig-11- write stats script

3.2.4 Dash client data collection: -

After installing all the required software's and applications and configure your dash client, now all needed is a client to open the browser and start streaming the video, and dash automatically will start collecting bitrate, throughput and buffer from the client and from those information's the system will decide to move from one node to another one, with prefetching video segments in the upcoming node for assuring better QoE to the end user, fig-12- shows client watching a video and his data being collected.

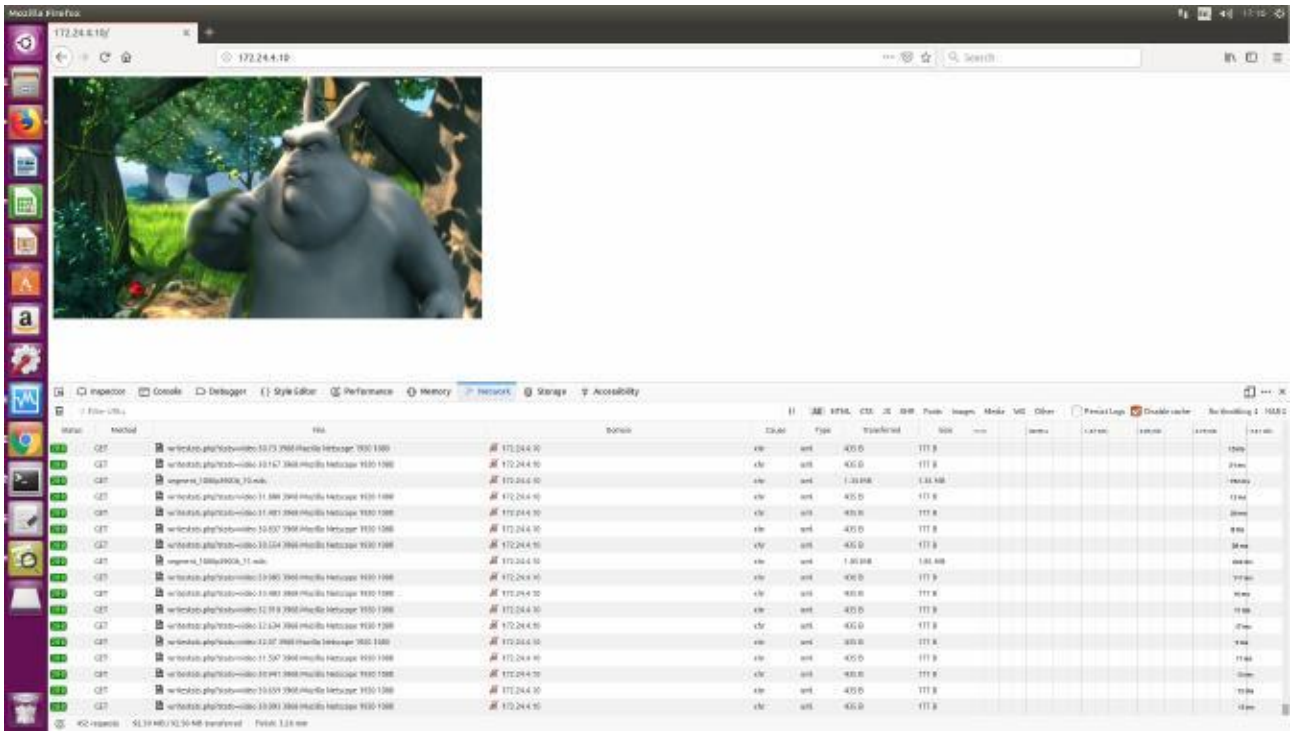


Fig-12- Dash client while data being collected

3.3 Testbed architecture

To illustrate the effect of the edge-based video streaming and mobility, Ubuntu 16.04 LTS desktop was used together with two laptops. The Desktop was used to simulate a cloud environment using Devstack (Open-Stack), which included: network, storage, compute and controller in the same node. An instance of Ubuntu cloud operating system was launched which had minimal Ubuntu image to host the MPEG-DASH video server. For DASH streaming in the cloud, Dash.js was used, it is a Javascript Based dash client that has been set as the reference client by Dash industry forum. Big Buck Bunny video sequence was used as the source video. The length of the video was 7:58 minutes compressed under different representations using h264 codec as shown in Table I. Apache was used as the Web Server in the cloud. The IP of the instance was taken from a private IP range pool. Since the Devstack instance had a private IP address, it was necessary to make interaction with the Public network by defining the static routes in the router.

Video Rep.	Bitrates
1080 P	3900k,3300k,2400k
720 P	2000k,1500k,1200k
480 p	700k,600k
360 P	500k,400k

TABLE 2 VIDEO REPRESENTATION

The edge nodes were based on the Ubuntu 16.04 virtual machines on the laptops. The Ubuntu virtual machines were configured with LXD containers. The LXD containers are lightweight and support live migration. For the testing purposes and to show a proof of concept, two edge nodes were used. The Squid software was installed on the LXD containers to act as the Web reverse proxy servers to the back-end cloud MPEG-DASH video server. Fig.13 depicts the overall testbed for this paper. Fig. 13. Testbed setup for live migration to emulate network varying conditions between user mobiles and edge nodes, Netem tool was used inside the containers. The network bandwidth was varied between
512 Kbps and 2 Mbps. These variations were enough to cause video quality fluctuations.

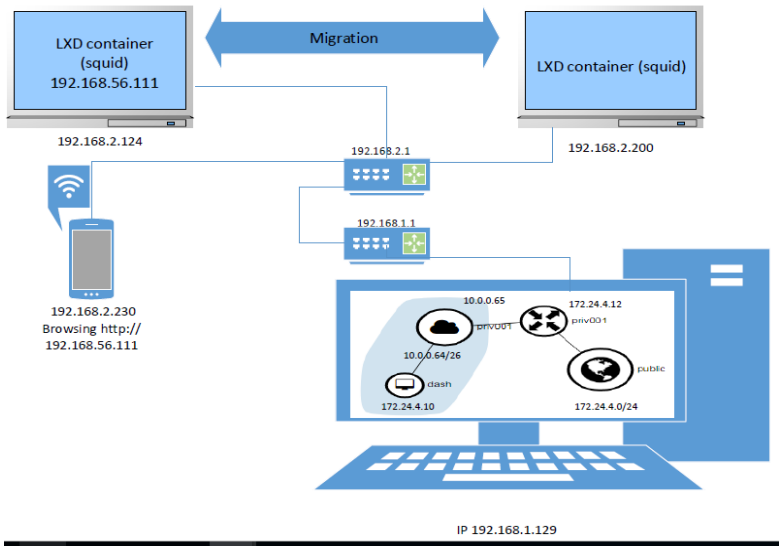


Fig -13- Testbed setup

The proposed system architecture for the ENP scheme is illustrated in Fig.1. The MPEG-DASH server is hosted in the cloud and the mobile edge nodes are within the proximity of the ENodeBs. The edge nodes will act as reverse Web proxies for the user mobile clients on behalf of the MPEG-DASH server in the cloud. During streaming, if the video segments are not cached in the edge node then the edge node will request these segments from the cloud, cache them and serve the mobile clients. If the video segments are cached in the edge node then the edge node will serve the mobile clients without requesting them from the cloud. In this system, the mobile clients also communicate directly with the MPEG-DASH by transmitting some computed network parameters (e.g., throughput, current client buffer size, segment number and bitrate) to the MPEG-DASH server.
When a mobile client moves e.g. from left to right as shown in Fig.1, it will cross over many different mobile edge nodes along the path. To avoid video quality drop during video streaming from one node to another, an appropriate amount of video segments will be prefetched in following-on nodes, in advance, along the path in a follow-me prefetching manner. This is to make sure that video streaming services are seamless, and the QoE for streaming services provided to the end user is well maintained during user mobility states (e.g. watching streaming video in a moving car).

4- Results and Performance Comparison

During the experiment, the impacts of buffer size and bitrate on video quality were investigated. Several tests were conducted to determine the predetermined client buffer size threshold for migration. For the dash.js clients and video segment sizes used in the experiments, the maximum client buffer size was always at around 32 ms. Through empirical results the predetermined client buffer threshold size of 20 ms was set for migration, to determine this threshold different tests were carried out under different thresholds, starting the migration process before 20ms was not effecting any parameters and the video was continually playing at the new node without any drop down of the quality, and starting the migration process after the threshold was causing buffer drop down for 0 and the DASH client was not able to recover 60% of the time in some approaches when Netem tool were used.

Three schemes (ENP, Container Wakeup (C-up), Live Migration [15]) were compared under different bandwidth limitations, ENP and C-UP scheme are in the section above with details, the only difference between the two proposed approaches are the prefetching part in ENP which does not exist in C-up, The Live Migration is from liturature, . It was found that the live migration scheme had the largest delay compared to ENP and C-up schemes. This is because live migration scheme moves the whole container, and it will use checkpoint/restore to save the status of all the processes in the container and start them again in the new node, while the ENP scheme only wakes up the new container and prefetches a few segments to the new edge node. The C-up scheme experiences the least delay because it only wakes up the new container without prefetching. Fig-14- illustrates the execution time of each aforementioned scheme.

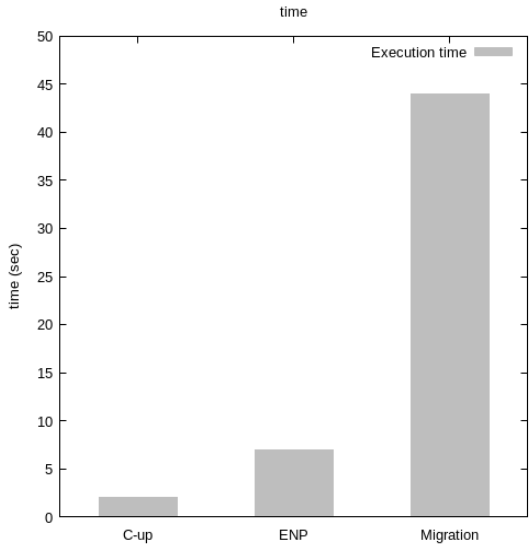


Fig -14- execution time for all approaches

Extensive tests have been carried out to determine the most efficient among all the three approaches to optimise QoE for the end user. We ran the test 10 times for each approach, and as expected the live migration takes the longest time to be executed as compared to the other two approaches. The increase in time period occurs because the whole container is moved from one node to another so it will take time to save the status of all process, then stop the container, move it to the new node, and then start it in the new node and restore all the process. The IPchange approach takes the minimum time to be executed, and the IPchange with

prefetching approach takes longer time to be executed than the IPchange one but it is still much faster than the Live migration approach.

The relationship of the bitrate levels vs. time is shown in Fig-15-, as anticipated, because of the long execution time of Live migration scheme. It can be seen clearly the drop of the bitrate levels. During the live migration the video quality drops to the minimum quality available. For the C-up scheme the video quality will drop to the medium quality for a short time then regain the original high quality. For the ENP scheme, there was no change in the video quality because of the video segments were prefetched in the target edge node before stopping the old edge node.

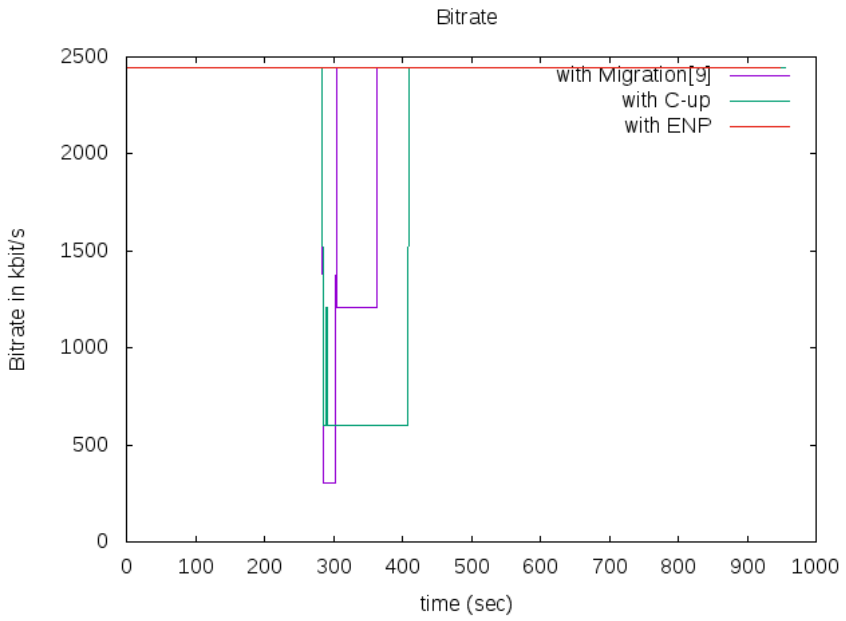


Fig-15- bitrate level

Fig-16- illustrates how buffer size changes vs. time. It can be seen that the live migration buffer size drops the most compared to the others, because it takes the longer execution time for saving the status of the edge nodes and restoring all the processes after migration. The ENP scheme which executes the new edge node waking up and prefetching video segments experienced the least drop in the buffer level, this is because the execution time was better than that of the live migration and also due to the positive impact of the prefetching. The C-up scheme buffer level was better than that of the live migration scheme, this is because the execution time was better than the live migration approach.

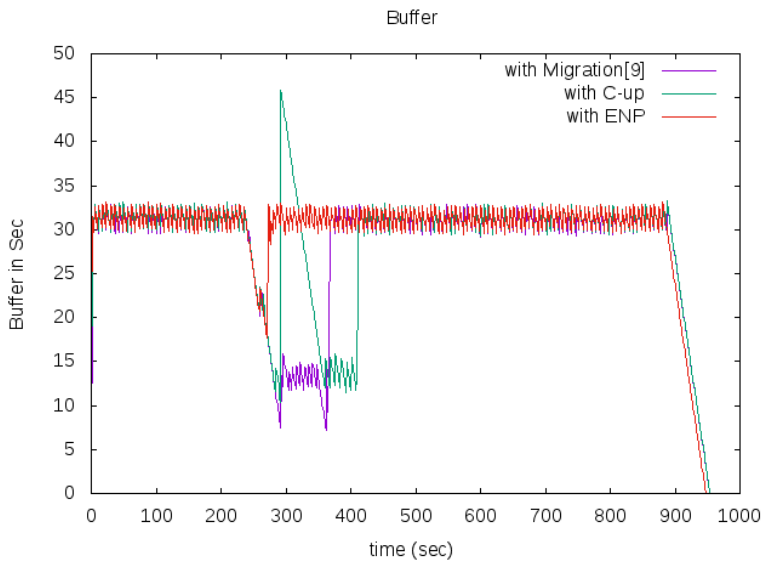


Fig-16- buffer levels

We further tested the system under different network conditions by emulating different network packet loss settings via Netem. 10 tests were carried out for each approach to show the average bitrate under different packet loss conditions. We observe that no impacts are shown in the video's bitrate on all the approaches when the packet loss ratio is between 5%-15%, and when the ratio is increased to 30% or more the Live Migration approach witnesses a huge amount of bitrate drops, and sometimes not recovering at all after releasing the packet loss condition

To overcome the not recovering stats of dash we started to use Iperf instate of Netem to play with the network condition and put load on the network, again 10 tests were carried out for each approach to show the average bitrate received for each approach, fig -17- shows the Average bitrate interval for each approach.

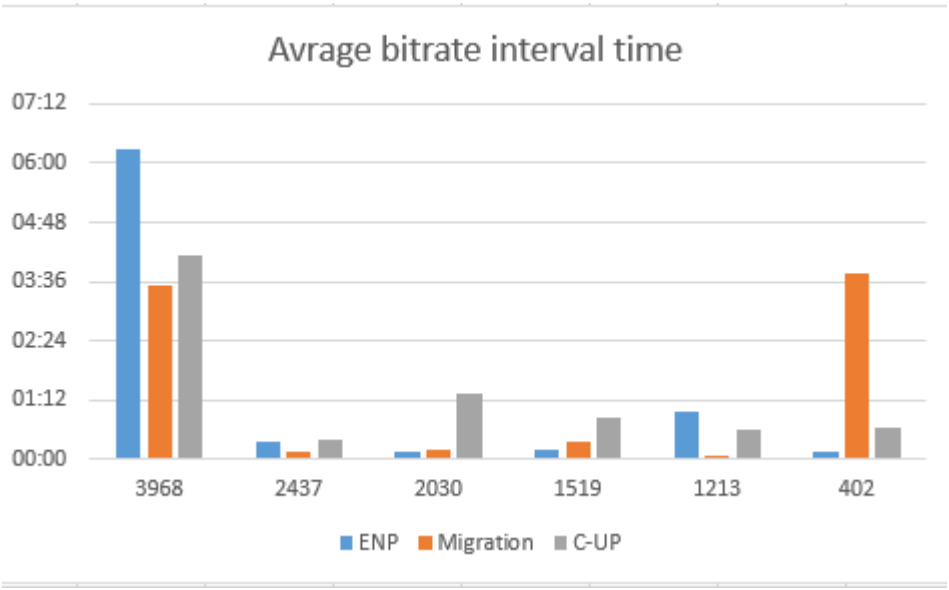


fig -17- Average bitrate interval

as indicated in Fig-17- that ENP scheme maintains higher bitrate levels comparing to Live migration, and C-up, and that relates to the long time that Live migration take to use the checkpoint/restro and save the stats of all the process then move the whole container from one node to the other then restore the stats of all process. C-Up and ENP has close stats but ENP takes the advantage of prefetching at the new node and recovers faster so we see less drop down in the quality for ENP scheme.

Fig-18- shows the average switching for each approach, in case of bitrate switching, it is obvious that C-Up scheme will have the most number of switching almost every time because of the quick drop down in the network and the slow recovery in the new node. ENP in this case will take the advantage of prefetching to maintain a reasonable number of switching again, while Live migration approach has the less number of switching because of the long time it takes to move a container from the old node to the new one, and fig -19- shows the number of stats received for each bitrate in each approach.

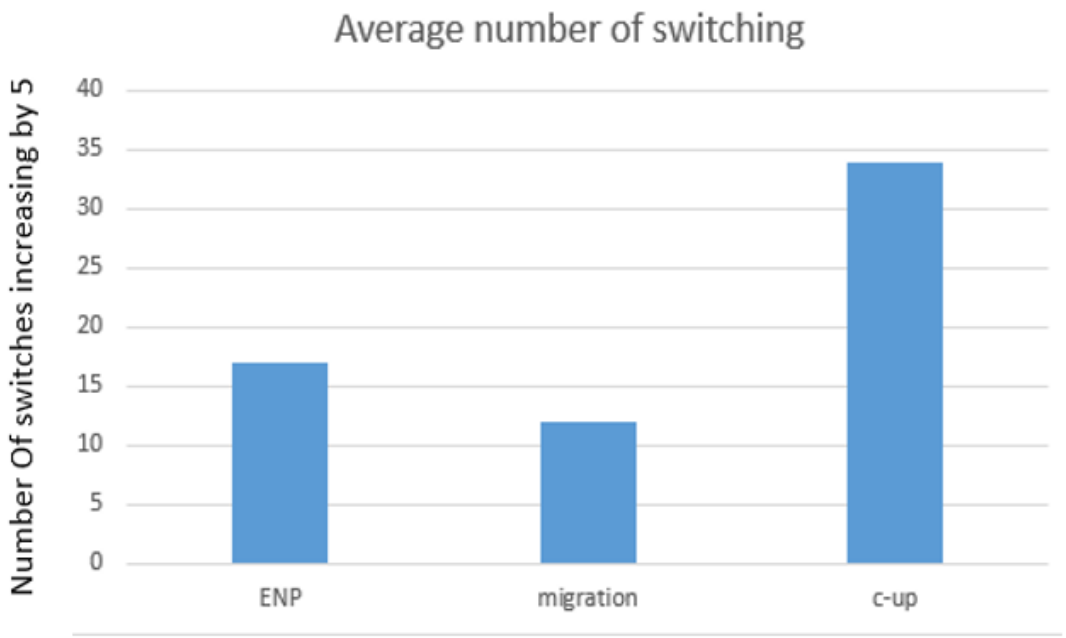


Fig-18- Average number of switching

In ENP scheme average bitrate received is the highest one in every test we run because of the prefetching advantage, while Live migration approach suffers from drop down in bitrate when moving from one node to another due to the time taken by the process, so as expected the Live migration will be getting the lowest bitrate available of the longest time compared to other approaches, C-up shows similar results to ENP scheme apart from the movement time when the bitrate might drop for a lower quality for a short time then recovers to the highest one.

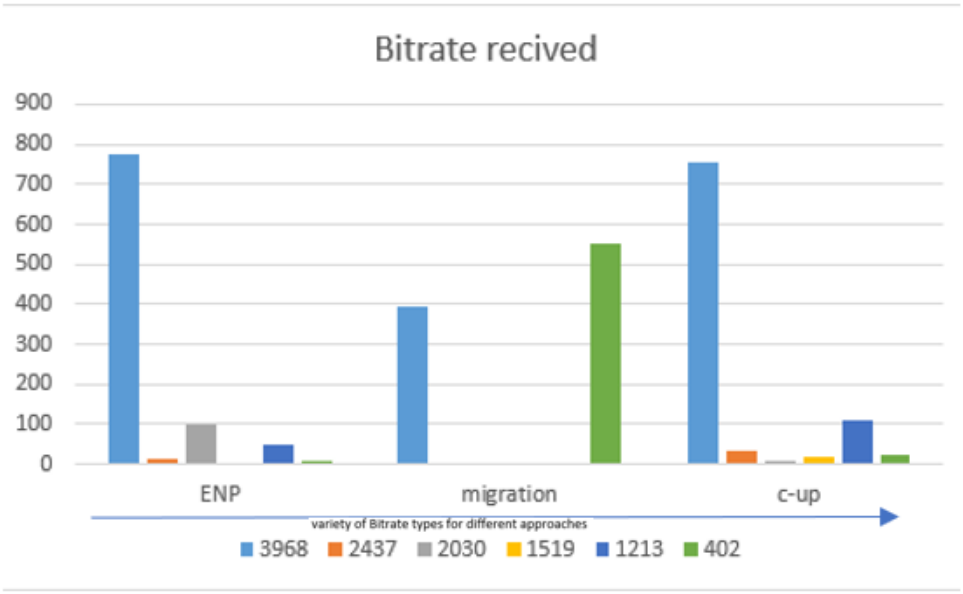


Fig-19- Amount of bitrate received for each approach

In most of the tests for live migration approach when the movement was taking a bit longer than expected the dash was unable to recover and the buffer level was drop to 0 and staying their until a new session starts by applying the ENP scheme we can avoid that event as well, fig-20-21 shows a buffer level, and bitrate for an unrecovered Live migration test.

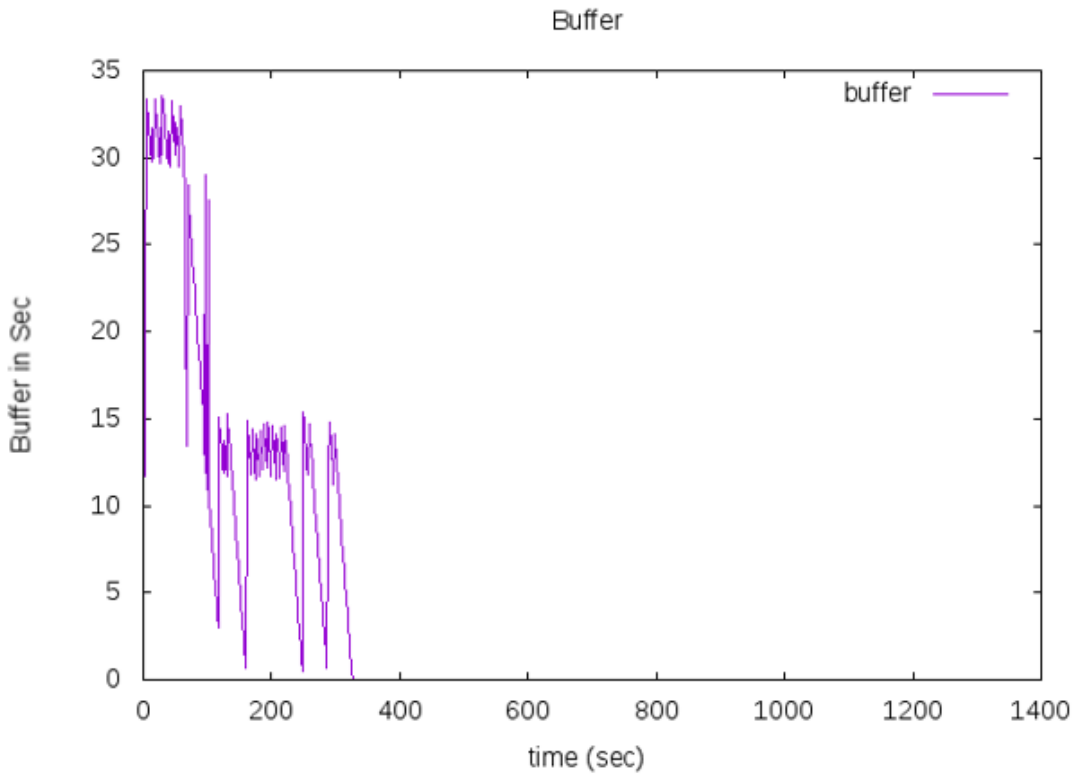


Fig-20- unrecovered buffer level

figure-20- describes a common unrecoverable drop down in the buffer levels due to live migration approach execution time, when the live migration execution time exceeds the normal levels and buffer level reaches zero, even if the container got migrated successfully to the new node, the system will be unable to recover, that gives ENP approach one more advantages which is avoiding such kind of drop downs.

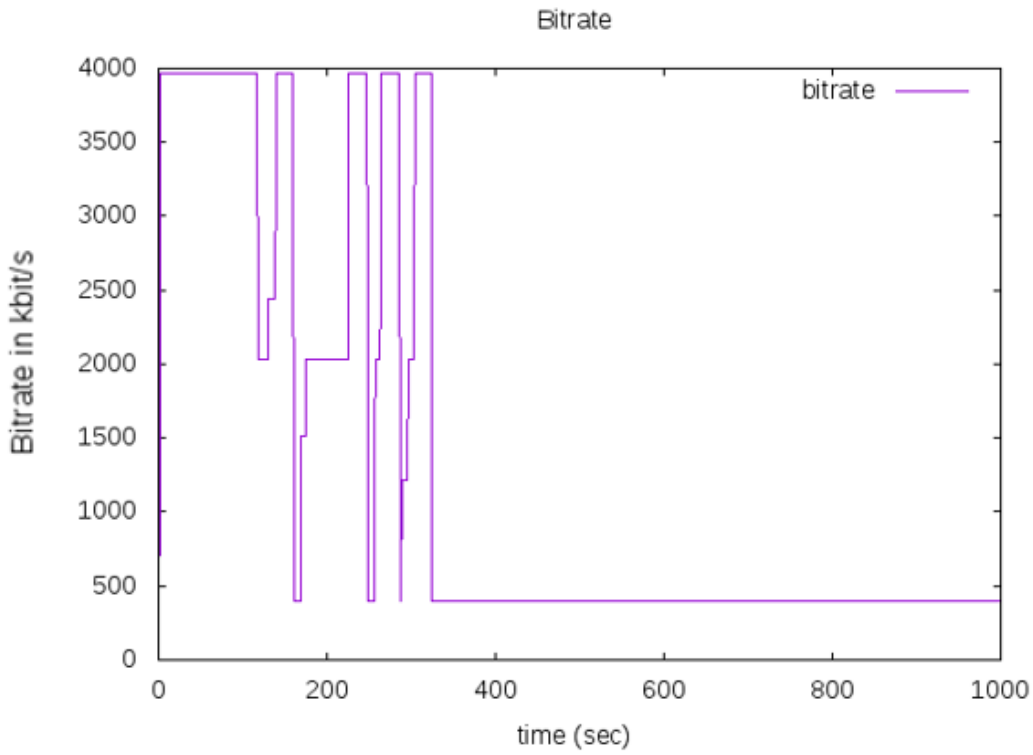


Fig-21- unrecovered bitrate stats

Figure-21- shows the unrecoverable bitrate stats for live migration approach when the execution and checkpoint/restro functions takes abnormal times to be executed.

5- Conclusions and Future Work

5.1 Conclusions

In this project, a study was carried out to enhance video streaming quality for mobile video streaming applications. The key contributions are summarized as below:

1. A testbed was setup to assess the quality of mobile video streaming services based on DASH.
2. novel follow-me Edge Node Prefetching (ENP) scheme that wakes up the target edge node and prefetches an appropriate amount of video segments in advance in order to avoid video quality degradation during service migration in mobile edge nodes. The testbed based on the OpenStack, two edge nodes (LXD Containers) and a mobile device was set up and used to obtain empirical results. The MPEGDASH scheme based on dash.js was utilised to assess and evaluate the performance of the proposed scheme and compared with that of the existing approaches. Preliminary results have shown that the ENP scheme can achieve better video quality and less service migration time than that of live migration and C-up schemes.

5.2 Limitations and Future Work

Due to time constraints, the testbed and experiment carried out was limited, and the proposed approach, e.g. when to wake up and/or move the container was fixed and not flexible. Future work can be done in the following two directions, e.g.

1. More intelligent approach for containers wake-up/movement for seamless mobile video streaming services.
2. An appropriate Quality of Experience (QoE) assessment for mobile video streaming services. This could involve an appropriate subjective assessment method or utilise the latest objective quality assessment methods for mobile video streaming services to compare the performance between the proposed approach and existing ones.

References

1- Cisco. (2017, Sep) Cisco Visual Networking Index:Forecast and Methodology, 2016-2021. [Online].
Available:https://www.cisco.com/c/en/us/solutions/collateral/serviceprovider/visual-networking-index-vni/complete-white-paper-c11-481360.html

2- B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia Streaming via TCP: An Analytic Performance Study", *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 4, no. 2, May 2008, pp. 16:1-16:22.

3- H. Schwarz, D. Marpe, T. Wiegland, "Overview of the scalable video coding extension of the H.264/AVC standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol 17, no. 9, Sep. 2007, pp. 1103-1120.

4- 3GPP TS 26.234, "Transparent end-to-end packet switched streaming service (PSS)", Protocols and codecs, 2010.

5- S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," IEEE Access, vol. 5, pp. 6757–6779, 2017.

6- R. Pantos, W. May, "HTTP Live Streaming", IETF draft, March. 2011, http://tools.ietf.org/html/draft-pantos-http-live-streaming-06

7- Adobe HTTP Dynamic Streaming, http://www.adobe.com/products/httpdynamicstreaming/

8- Microsoft Smooth Streaming, http://www.iis.net/download/smoothstreaming

9- C. Müller, C. Timmerer, "A Test-Bed for the Dynamic Adaptive Streaming over HTTP featuring Session Mobility", *ACM Multimedia Systems*, San Jose, California, USA, Feb. 2011, pp. 271-276.

10- M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Nealet al., "Mobile-edge computing introductory technical whitepaper," White Paper, Mobile-edge Computing (MEC) industryinitiative, 2014.

11- Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young,"Mobile edge computinga key technology towards 5g,"ETSIwhite paper, vol. 11, no. 11, pp. 1–16, 2015.

12- H. Chang, A. Hari, S. Mukherjee, and T. Lakshman, "Bringing the cloud to the edge," in Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on. IEEE, 2014, pp. 346–351.

13- J. O. Fajardo, I. Taboada, and F. Liberal, "Improving content delivery efficiency through multi-layer mobile edge adaptation," IEEE Network, vol. 29, no. 6, pp. 40–46, 2015.

14- Y. Jararweh, L. Tawalbeh, F. Ababneh, and F. Dosari, "Resource efficient mobile computing using cloudlet infrastructure," in Mobile Ad-hoc and Sensor Networks (MSN), 2013 IEEE Ninth Int. Conf. on. IEEE, 2013, pp. 373–377.

15- T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," IEEE Communications Magazine, vol. 55, no. 3, pp. 38–43, 2017.

16- J. W. Kleinrouweler, S. Cabrero, and P. Cesar, "Delivering stable high-quality video: An sdn architecture with dash assisting network elements," in Proceedings of the 7th Int. Conf. on Multimedia Systems. ACM, 2016, p. 4.

17- S. Chen, B. Shen, S. Wee, and X. Zhang, “Segment-based streaming media proxy: modeling and optimization,” IEEE Transactions on Multimedia, vol. 8, no. 2, pp. 243–256, 2006.
18- C. Ge, N. Wang, G. Foster, and M. Wilson, “Towards qoe-assured 4k video-on-demand delivery through mobile edge virtualization with adaptive prefetching,” IEEE Tran. on Multimedia, vol. 19, no. 10, pp. 2222–2237, 2017.
19- V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri, “Bandwidth-aware prefetching for proactive multivideo preloading and improved has performance,” in Proceedings of the 23rd ACM int. conf. on Multimedia. ACM, 2015, pp. 551–560.
20- J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston et al., “The youtube video recommendation system,” in Proc. of the 4th ACM.
21- Mahdi, A. E. (2007) voice quality measurement in modern telecommunication networks. 14th international workshop on system, signals and image processing,2007 and 6th EURASIP conference focused on speech and image processing, multimedia communication services. Pp25-32.
22- www.ffmpeg.org

Appendix -1- submitted paper

Follow-me Prefetching for Video Streaming over Mobile Edge Computing Networks

Ibrahim S. Mohammedameen, Is-Haka Mkwawa and Lingfen Sun

School of Computing, Electronics and Mathematics
Plymouth University, Plymouth PL4 8AA, UK

E-mail: {Ibrahim.mohammedameen; Is-haka.mkwawa; L.Sun}@plymouth.ac.uk

***Abstract*—Mobile video streaming services have increased exponentially in recent years due to the popularity of mobile devices, the advancement of mobile networks, and the availability of a variety of video contents over the Inter- net. Mobile Edge Computing (MEC), in connection with the backend Cloud computing, has been used to bring contents close to the end user in order to reduce transmission latency. However, the quality of video streaming services suffers from degradation when an end user moves from the coverage of one node to another or when the condition of a mobile network degrades. In this paper, we propose a novel follow-me Edge Node Prefetching (ENP) scheme to prefetch appropriate video segments in advance in the following- on mobile node to avoid video quality degradation during video streaming. We set up a test bed consisting of a back- end cloud (OpenStack), two edge nodes (LXD Containers) and a mobile device, and implemented the ENP algorithms on cloud server and client sides. Extensive experiments for Dynamic Adaptive video Streaming over HTTP (DASH) services were carried out based on dash.js from the DASH Industry Forum. Preliminary results show that the ENP scheme can achieve better video quality (in terms of provisioning of average video bit rate per segment) and less service migration time between mobile nodes when compared with existing approaches. The scheme might be useful in supporting video streaming services over MEC, and/or in future smart city applications.**

***Index Terms*—QoE, MEC, MPEG-DASH, prefetching, Migration**

I. Introduction

With the advancement of mobile communications and video processing technologies in recent years, video streaming services (such as those provided by YouTube and Netflix) have become one of the most dominant services over the Internet. According to the latest Cisco Visual Networking Index (VNI), video services will account for about 82% of all consumer Internet traffic by 2021, growing threefold between 2016 to 2021 [1]; mobile video will be about 78% of the world's mobile data traffic by 2021, increasing 9-fold between 2016 and 2021 [1]. In this context, great efforts have been made

to improve the quality of video streaming services over the mobile Internet.

Over the years, Over-The-Top (OTT) service providers such as YouTube and Netflix have been using the MPEG- DASH (Dynamic Adaptive Streaming over HTTP) stan- dard to deliver video streaming services over the Internet [2]. MPEG-DASH has many advantages such as flexible quality adaptation and simplicity in implementation due to its easy integration with the existing HTTP-based infrastructure. In MPEG-DASH, the video is prepacked with short video segments for different representations corresponding to different video qualities at the server. A DASH client predicts available network bandwidth and sends an HTTP-request to the server for appropriate video representation segments. Due to varying network conditions, an ideal DASH scheme should be able to adapt between different video segments to provide ac-ceptable video quality and hence, acceptable Quality of Experience (QoE) to video streaming customers. Normally video contents are located in servers in Data Centres over the Cloud and/or in cache servers provided by Content Delivery Networks (CDN). When a client is too far away from the server (e.g. across a continent), latency and network congestion will have adverse effects on video streaming quality. The situation will get worse when the Internet video is streamed to mobile devices.

Mobile Edge Computing (MEC), a promising technol- ogy for 5G mobile networks, has been utilised to assist video streaming services due to its ability to facilitate the provisioning of high data rate and low latency services to end users and its ability to provide computational power at the mobile edge. It has been used for MPEG-DASH to reduce the backbone traffic to the Cloud and bring video contents close to the end user. The MEC specified by ETSI [3], is one of the Mobile Edge Network (MEN) structures that offers intense computing capability along with real-time communication ability with the end user.

The MEC has provided an ability to bring video

contents close to mobile users. However, with mobil- ity of users across different mobile nodes and varying mobile network conditions, maintaining and achieving high QoE for video streaming services remain a great challenge. In this paper, we propose a novel follow-me Edge Node Prefetching (ENP) scheme to prefetch some video segments in advance in following-on mobile nodes to avoid video quality degradation due to transition from one node to another. In this scheme, the user will be served from the new node with seamless interruption and assurance of high quality video delivery. We set up a test bed consisting of a back-end cloud (OpenStack), two edge nodes (LXD Containers) and a mobile device, and carried out extensive experiments for DASH services based on dash.js. Preliminary results show that the ENP scheme can achieve better video quality (in terms of provisioning of average video bit rate per segment) and less service migration time between mobile nodes when compared with existing approaches.

The proposed ENP scheme can be applied in several scenarios, such as,

1) the user's current edge node experiences bad network conditions (such as poor signal strength and low bandwidth). In this scenario, the user will be redirected to the next closest edge node to continue with video streaming with seamless interruption and video quality degradation.
2) the user is on the move from one mobile node to the next. In this case, the user will be redi- rected to the next node with better signal strength to continue with video streaming with seamless interruption.

In these scenarios, video segments are prefetched before the user is redirected to the next edge node. The main problem with the existing approaches are that they suffer from processing delays which affect video streaming quality at the end user.

The main objective of this study is to optimise QoE for the end user using MEC regardless of the mobility of the user. The approach ensures that the user is always served from the closest nodes with better network quality.

The main contributions of the paper are twofold:

- A novel follow-me Edge Node Prefetching (ENP) scheme. The scheme is able to prefetch video segments in advance in following-on mobile edge node to tackle the problem of video quality drop for video streaming services due to service migration.
- The proposed ENP algorithms on cloud and client sides. The algorithms have been implemented for video streaming services in a cloud/MEC integrated testbed using OpenStack, LXD Container and dash.js. The performance is evaluated and compared with the existing approaches, and better video streaming quality is obtained for the proposed scheme.

The rest of this paper is organised as follows. The related work is discussed in Section II. Section III presents the ENP scheme and its algorithms for the cloud server and the client sides. Section IV explains the experimental setup including the testbed used of all approaches. The results and discussion are summarized in Section V. Conclusions and future work are given in Section VI.

II. Related work

The delivery of mobile content to users is the main motive of network systems. At present, the delivery of high definition videos with high resolution has be- come the prime focus to enhance the context of 5G network development in the future. The 5G network enables increased network capacity along with the QoE by adding required network intelligence in all type of network requirements. The recent development of new network paradigms, Mobile Edge Computing (MEC) [4] in particular is anticipated that intelligence and video content awareness can be implemented in the edge node for optimizing the QoE for end users.

The principal objective of MEC is to place storage and computation resources at the network edge, in the user vicinity. The data processing can be driven accordingly from inaccessible cloud to the edge. When the data is processed locally and data streams are accelerated via several techniques such as caching and compression, MEC reduces the bottleneck toward the core network. In addition, it decreases end-to-end latency, enabling the offload of important computation load from power constrained user equipment to the edge. The executive briefing of the European Telecommunications Standards Institute (ETSI) MEC initiative argues that one edge computing shall enable new computation-intensive services and shall yield promising business models. It also represents a fault resilient solution for its decentralized architecture [5].

The research work in [6] uses network assisted adaptive streaming applications for multimedia content delivery inside MEC to enhance Quality of Experience (QoE). The research work in [7] makes use of edges as caches along with proxies to store media content.

Most of the MEC studies are focusing about reducing the latency, Taleb [8] tackles the migration problems in the context of smart cities. They proposed an approach

to enhance video streaming experience for the end user in smart cities based on Follow me Edge concept, unlike our studies they are focusing on the users mobility from one node to another and not on the video quality.

Several studies have been carried out on the concept of prefetching, and how to determine the right video to be prefetched to the user, without wasting resources. The study in [9] does neither include prefetching nor caching at the network edge, they only predict the network condition parameters and send them to the controller. Authors in [10] proposed a scheme which decides when and how many segments to prefetch, however, they did not consider varying network conditions. This could lead to the video quality drop because of segments downloading delays. Chang's work [11] showed the importance of content localization and how to reduce end-to-end delay when the video is on the global Internet, by proposing an approach called Mobile edge Virtualization with Adaptive Prefetching (MVP).This was done by checking RAN parameters and prefetch video segments into the edge node to maintain a progress gap from the users actual request , but they didnt mention/consider edge node changing or users mobility in their work.

Krishnamoorthi [12] focused on bandwidth-aware prefetching. Three approaches were proposed to prefetch the alternative videos that the user might watch after finishing the main one. Based on the bandwidth they predicted the maximum buffer size, after reaching the maximum buffer size the application will stop downloading the main video and start downloading the alternative ones. Other approaches [13] determined which video the user most likely to watch next, however, these approaches may sometimes prefetch unwanted videos which lead to resource wastage. In this paper, the pro- posed ENP scheme combines the prefetching with the migration approach to prefetch the video segments in a new edge node before that node starts to serve the user. This will make sure that video streaming services can be delivered to an end user without quality degration when serving mobile edge node is migrated from one to another.

III. Follow-me Edge Node Prefetching (ENP) Scheme

The proposed system architecture for the ENP scheme is illustrated in Fig.1. The MPEG-DASH server is hosted in the cloud and the mobile edge nodes are within the proximity of the ENodeBs. The edge nodes will act as reverse Web proxies for the user mobile clients on behalf of the MPEG-DASH server in the cloud. During streaming, if the video segments are not cached in the edge node then the edge node will request these segments from the cloud, cache them and serve the mobile clients. If the video segments are cached in the edge node then the edge node will serve the mobile clients without requesting them from the cloud. In this system, the mobile clients also communicate directly with the MPEG-DASH by transmitting some computed network parameters (e.g., throughput, current client buffer size, segment number and bitrate) to the MPEG-DASH server. When a mobile client moves e.g. from left to right as shown in Fig.1, it will cross over many different mobile edge nodes along the path. To avoid video quality drop during video streaming from one node to another, an appropriate amount of video segments will be prefetched in following-on nodes, in advance, along the path in a follow-me prefetching manner. This is to make sure that video streaming services are seamless, and the QoE for streaming services provided to the end user is well maintained during user mobility states (e.g. watching streaming video in a moving car).

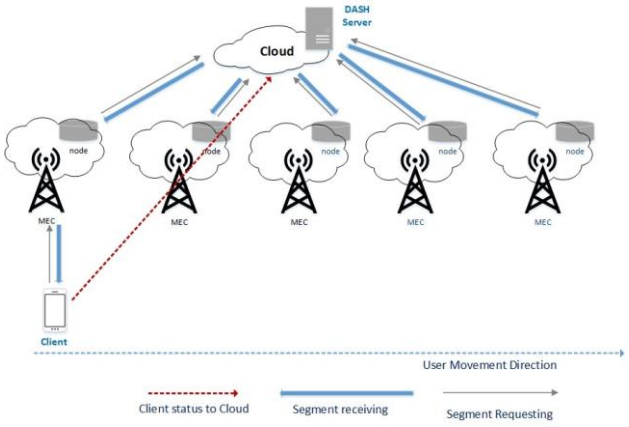


Fig. 1. The ENP system architecture

The proposed ENP scheme has two algorithms that run in the cloud server and mobile client side, respectively. The cloud side algorithm is listed as Algorithm 1, which starts by checking the client's buffer size, B_t (Buffer size at time t). The buffer size is sent periodically (here every 250 ms) from the client side (c.f., Algorithm 2), which fluctuates as a result of network conditions. When the current buffer size is less than the predetermined buffer size threshold (B_{th}), the cloud side algorithm will wake up the next closest edge node (here container $C_{(k+1)}$) and start prefetching the next predetermined number of segments (p_s = 5 segments in this example). After prefetching process has finished, the user will then be migrated to continue with streaming from the new edge

node, the old edge node (container C_k) will then be stopped.

On the client side, assuming the current serving video segment is S_i and buffer size at time t is B_t, for each segment, it will calculate/estimate the current buffer size (B_t), and then send the buffer size (B_t) and the segment number (S_i) to the cloud.

Algorithm 1: The Cloud side ENP algorithm

Let t= {0,....,T }
Δt= 250 ms
Current segment S_i
current container : C_k
let p_s be the number of segments to prefetch
B_t : Clients buffer size at time t
B_{th}: predefined buffer threshold **foreach** Δt **do**

obtain B_t

if $B_t < B_{th}$ **then**

wake up container $C_{(k+1)}$
for *(j=i; j <i +p_s; j++)* **do**

prefetch segment S_j in C_{k+1}
end

stop C_k

end;

Algorithm 2: The client side ENP algorithm

Current segment S_i
Client buffer size at time t, B_t
foreach S_i **do**

calculate B_t
send B_t to the cloud
send to the cloud
S_i

;

IV. Experimental setup

To illustrate the effect of the edge-based video streaming and mobility, Ubuntu 16.04 LTS desktop was used together with two laptops. The Desktop was used to simulate a cloud environment using Devstack (OpenStack), which included network, storage, compute and controller in the same node. An instance of Ubuntu cloud oper- ating system was launched which had minimal Ubuntu image to host the MPEG-DASH video server. For DASH streaming in the cloud, dash.js was used, which is a Javascript-Based dash client that has been set as the reference client by DASH Industry Forum. Big Buck Bunny video sequence was used as the source video. The length of the video was 7:58 minutes compressed under different representations using H.264 codec as shown in Table I. Apache was used as the Web Server in the cloud. The IP of the instance was taken from a private IP range pool. Since the Devstack instance had a private IP address, it was necessary to make interaction with the public network by defining the static routes in the router.

TABLE I

VIDEO REPRESENTATION

Video Rep.	Bitrates
1080 P	3900k,3300k,2400k
720 P	2000k,1500k,1200k
480 P	700k,600k
360 P	500k,400k

The edge nodes were based on the Ubuntu 16.04 virtual machines on the laptops. The Ubuntu virtual ma- chines were configured with LXD containers. The LXD containers are lightweight and support live migration. For the testing purposes and for a proof of concept, two edge nodes were used. The Squid software was installed on the LXD containers to act as the Web reverse proxy servers to the back-end cloud MPEG-DASH video server. Fig. 2 depicts the overall testbed used.

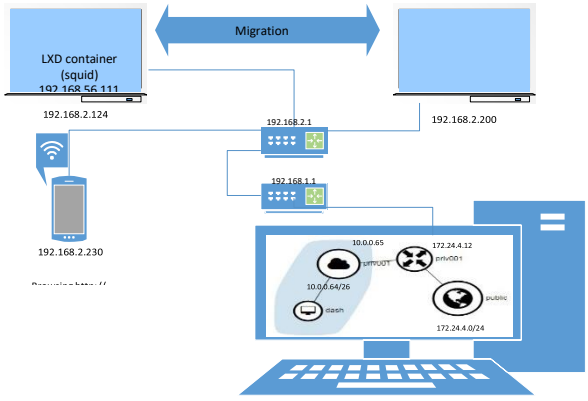


Fig. 2. Testbed setup for live migration

To emulate network varying conditions between user mobiles and edge nodes, Netem tool was used inside the containers to emulate different network bandwidth or different network packet loss conditions. The network bandwidth was varied between 512 Kbps and 2 Mbps with buffer size of 1600, 3200, or 6400 bytes. For the packet loss tests, all approaches were tested under different packet loss conditions between 5 -30% packet

loss, where packet loss was introduced to the network at a fixed time for 20 seconds then released. These network variations were sufficient to cause video quality fluctuations.

V. Results and Discussion

During the experiment, the impacts of buffer size and bitrate on video quality were investigated. Several tests were conducted to determine the predetermined client buffer size threshold for migration. For the dash.js clients and video segment sizes used in the experiments, the maximum client buffer size was always at around 32 ms. Through empirical results the predetermined client buffer threshold size of 20 ms was set for migration.

Three schemes (ENP, Container Wakeup (C-up), Live Migration [8]) were compared under different bandwidth limitations. Both the ENP and the C-up are proposed by the authors and the differenc between them is that C-up does not do prefetching. The Live Migration is from liturature. It was found that the live migration scheme had the largest delay compared to ENP and C-up schemes. This is because live migration scheme moves the whole container, and it will use checkpoint/restore to save the status of all the processes in the container and start them again in the new node, while the ENP scheme only wakes up the new container and prefetches a few segments to the new edge node. The C-up scheme experiences the least delay because it only wakes up the new container without prefetching. Fig. 3 illustrates the execution time of each aforementioned scheme.

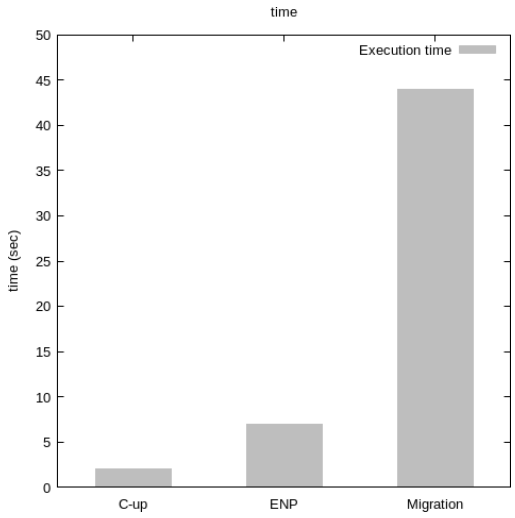


Fig. 3. Execution Time

The relationship of the bitrate levels vs. time is shown in Fig. 4, as anticipated, because of the long execution time of Live migration scheme. It can be seen clearly the drop of the bitrate levels. During the live migration the video quality drops to the minimum quality available. For the C-up scheme the video quality will drop to the medium quality for a short time then regain the original high quality. For the ENP scheme, there was no change in the video quality because of the video segments were prefetched in the target edge node before stopping the old edge node.

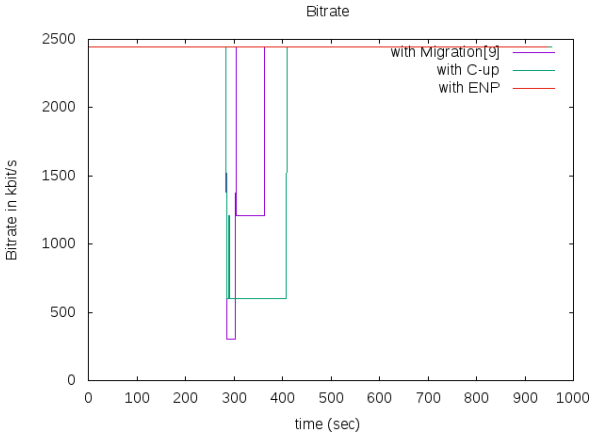
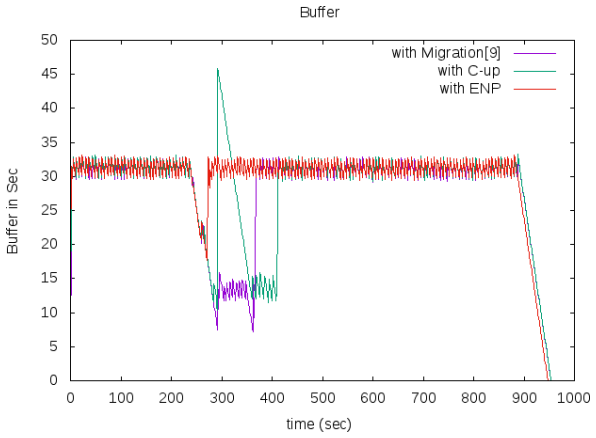


Fig. 4. Bitrate levels vs Time

Fig.5 illustrates how buffer size changes vs. time. It can be seen that the live migration buffer size drops the most compared to the others, because it takes the longer execution time for saving the status of the edge nodes and restoring all the processes after migration. The ENP scheme which executes the new edge node waking up and prefetching video segments experienced the least drop in the buffer level, this is because the execution time was better than that of the live migration and also due to the positive impact of the prefetching. The C- up scheme buffer level was better than that of the live migration scheme, this is because the execution time was better than the live migration approach.

We further tested the system under different network conditions by emulating different network packet loss settings via Netem. 10 tests were carried out for each approach to show the average bitrate under different packet loss conditions. We observe that no impacts are shown in the videos bitrate on all the approaches when the packet loss ratio is between 5%-15%, and when the ratio is increased to 30% or more the Live Migration approach witnesses a huge amount of bitrate drops,



achieve better video quality and less service migration time than that of live migration and C-up schemes.

The future work will enhance the proposed ENP scheme with more intelligence to adapt to different network conditions and different video contents in order to maintain better QoE for video streaming services. The QoE metrics, in addition to stalling events, initial/average buffering time will also be assessed. More edge nodes and different mobility patterns will be considered. The applications of the ENP scheme in smart city will also beexplored.

Fig. 5. Buffer levels vs Time

and sometimes not recovering at all after releasing the packet loss condition. Table II shows the average bitrate per segment adopted for each approach under different packet loss ratio.

TABLE II

AVRAGE VIDEO BITRATE ADOPTATION PER SEGMENT

Approaches	5-10%	15%	20%	30%
ENP	2400P	2400P	2400p	2400P
C-UP	2400P	2400P	1200p	700P
Live Migration	2400P	1200P	700p	400P

Based on the obtained results, it can be observed that in spite of the live migration approach being dynamic, it wont ensure high level of video quality to the end user, at the same time the ENP scheme is much reliable than both live migration and C-up schemes. The ENP scheme is characterized by less resource wasting and maintaining high video quality level for end users.

VI. CONCLUSION

This paper has proposed a novel follow-me Edge Node Prefetching (ENP) scheme that wakes up the target edge node and prefetches an appropriate amount of video segments in advance in order to avoid video quality degradation during service migration in mobile edge nodes. The testbed based on the OpenStack, two edge nodes (LXD Containers) and a mobile device was set up and used to obtain empirical results. The MPEG- DASH scheme based on dash.js was utilised to assess and evaluate the performance of the proposed scheme and compared with that of the existing approaches. Preliminary results have shown that the ENP scheme can

REFERENCES

[1] Cisco. (2017, Sep) Cisco Visual Networking Index: Forecast and Methodology, 2016-2021. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service- provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757– 6779, 2017.

[3] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[5] H. Chang, A. Hari, S. Mukherjee, and T. Lakshman, "Bringing the cloud to the edge," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. IEEE, 2014, pp. 346–351.

[6] J. O. Fajardo, I. Taboada, and F. Liberal, "Improving content delivery efficiency through multi-layer mobile edge adaptation," *IEEE Network*, vol. 29, no. 6, pp. 40–46, 2015.

[7] Y. Jararweh, L. Tawalbeh, F. Ababneh, and F. Dosari, "Resource efficient mobile computing using cloudlet infrastructure," in *Mobile Ad-hoc and Sensor Networks (MSN), 2013 IEEE Ninth Int. Conf. on*. IEEE, 2013, pp. 373–377.

[8] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mo- bile edge computing potential in making cities smarter," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 38–43, 2017.

[9] J. W. Kleinrouweler, S. Cabrero, and P. Cesar, "Delivering stable high-quality video: An sdn architecture with dash assisting network elements," in *Proceedings of the 7th Int. Conf. on Multimedia Systems*. ACM, 2016, p. 4.

[10] S. Chen, B. Shen, S. Wee, and X. Zhang, "Segment-based stream- ing media proxy: modeling and optimization," *IEEE Transactions on Multimedia*, vol. 8, no. 2, pp. 243–256, 2006.

[11] C. Ge, N. Wang, G. Foster, and M. Wilson, "Towardsqoe-assured 4k video-on-demand delivery through mobile edge virtualization with adaptive prefetching," *IEEE Tran. on Multimedia*, vol. 19, no. 10, pp. 2222–2237, 2017.

[12] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri, "Bandwidth-aware prefetching for proactive multi- video preloading and improved has performance," in *Proceedings of the 23rd ACM int. conf. on Multimedia*. ACM, 2015, pp. 551– 560.

[13] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston *et al.*, "The youtube video recommendation system," in *Proc. of the 4th ACM conf. on Recommender systems*. ACM, 2010, pp. 293–296.

Appendix -2- squid configuration

```
#        WELCOME TO SQUID 2.7.STABLE6
#        ----------------------------
#
#        This is the default Squid configuration file. You may wish
#        to look at the Squid home page (http://www.squid-cache.org/)
#        for the FAQ and other documentation.
#
#        The default Squid config file shows what the defaults for
#        various options happen to be.  If you don't need to change the
#        default, you shouldn't uncomment the line.  Doing so may cause
#        run-time problems.  In some cases "none" refers to no default
#        setting at all, while in other cases it refers to a valid
#        option - the comments for that keyword indicate if this is the
#        case.
#


# Configuration options can be included using the "include" directive.
# Include takes a list of files to include. Quoting and wildcards is
# supported.
#
# For example,
#
# include /path/to/included/file/squid.acl.config
#
# Includes can be nested up to a hard-coded depth of 16 levels.
# This arbitrary restriction is to prevent recursive include references
# from causing Squid entering an infinite loop whilst trying to load
# configuration files.
```

```
# OPTIONS FOR AUTHENTICATION
# -----------------------------------------------------------------------------

#  TAG: auth_param
#       This is used to define parameters for the various authentication
#       schemes supported by Squid.
#
#       format: auth_param scheme parameter [setting]
#
#       The order in which authentication schemes are presented to the client is
#       dependent on the order the scheme first appears in config file. IE
#       has a bug (it's not RFC 2617 compliant) in that it will use the basic
#       scheme if basic is the first entry presented, even if more secure
#       schemes are presented. For now use the order in the recommended
#       settings section below. If other browsers have difficulties (don't
#       recognize the schemes offered even if you are using basic) either
#       put basic first, or disable the other schemes (by commenting out their
#       program entry).
#
#       Once an authentication scheme is fully configured, it can only be
#       shutdown by shutting squid down and restarting. Changes can be made on
#       the fly and activated with a reconfigure. I.E. You can change to a
#       different helper, but not unconfigure the helper completely.
#
#       Please note that while this directive defines how Squid processes
#       authentication it does not automatically activate authentication.
#       To use authentication you must in addition make use of ACLs based
#       on login name in http_access (proxy_auth, proxy_auth_regex or
#       external with %LOGIN used in the format tag). The browser will be
#       challenged for authentication on the first such acl encountered
```

```
#        in http_access processing and will also be re-challenged for new
#        login credentials if the request is being denied by a proxy_auth
#        type acl.
#
#        WARNING: authentication can't be used in a transparently intercepting
#        proxy as the client then thinks it is talking to an origin server and
#        not the proxy. This is a limitation of bending the TCP/IP protocol to
#        transparently intercepting port 80, not a limitation in Squid.
#
#        === Parameters for the basic scheme follow. ===
#
#        "program" cmdline
#        Specify the command for the external authenticator.  Such a program
#        reads a line containing "username password" and replies "OK" or
#        "ERR" in an endless loop. "ERR" responses may optionally be followed
#        by a error description available as %m in the returned error page.
#
#        By default, the basic authentication scheme is not used unless a
#        program is specified.
#
#        If you want to use the traditional proxy authentication, jump over to
#        the helpers/basic_auth/NCSA directory and type:
#                % make
#                % make install
#
#        Then, set this line to something like
#
#        auth_param basic program /usr/local/squid/libexec/ncsa_auth /usr/local/squid/etc/passwd
#
```



```
#        "children" numberofchildren
#        The number of authenticator processes to spawn. If you start too few
```

```
#	squid will have to wait for them to process a backlog of credential
#	verifications, slowing it down. When credential verifications are
#	done via a (slow) network you are likely to need lots of
#	authenticator processes.
#	auth_param basic children 5
#
#	"concurrency" numberofconcurrentrequests
#	The number of concurrent requests/channels the helper supports.
#	Changes the protocol used to include a channel number first on
#	the request/response line, allowing multiple requests to be sent
#	to the same helper in parallell without wating for the response.
#	Must not be set unless it's known the helper supports this.
#
#	"realm" realmstring
#	Specifies the realm name which is to be reported to the client for
#	the basic proxy authentication scheme (part of the text the user
#	will see when prompted their username and password).
#	auth_param basic realm Squid proxy-caching web server
#
#	"credentialsttl" timetolive
#	Specifies how long squid assumes an externally validated
#	username:password pair is valid for - in other words how often the
#	helper program is called for that user. Set this low to force
#	revalidation with short lived passwords.  Note that setting this high
#	does not impact your susceptibility to replay attacks unless you are
#	using an one-time password system (such as SecureID). If you are using
#	such a system, you will be vulnerable to replay attacks unless you
#	also use the max_user_ip ACL in an http_access rule.
#	auth_param basic credentialsttl 2 hours
# Recommended minimum configuration:
#
```

```
# Example rule allowing access from your local networks.
# Adapt to list your (internal) IP networks from where browsing
# should be allowed

# Auth
auth_param basic program /usr/lib64/squid/basic_ncsa_auth /etc/squid/squid_passwd
acl ncsa_users proxy_auth REQUIRED
http_access allow ncsa_users


acl all src all
acl manager proto cache_object
acl localhost src 127.0.0.1/32
acl to_localhost dst 127.0.0.0/8 0.0.0.0/32

acl localnet src 10.0.0.0/8          # RFC1918 possible internal network
acl localnet src 172.16.0.0/12       # RFC1918 possible internal network
acl localnet src 192.168.0.0/16      # RFC1918 possible internal network
acl localnet src fc00::/7       # RFC 4193 local private network range
acl localnet src fe80::/10      # RFC 4291 link-local (directly plugged) machines

acl SSL_ports port 443
acl Safe_ports port 80               # http
acl Safe_ports port 21               # ftp
acl Safe_ports port 443              # https
acl Safe_ports port 70               # gopher
acl Safe_ports port 210              # wais
acl Safe_ports port 1025-65535 # unregistered ports
acl Safe_ports port 280              # http-mgmt
acl Safe_ports port 488              # gss-http
```

```
acl Safe_ports port 591		# filemaker
acl Safe_ports port 777		# multiling http
acl CONNECT method CONNECT

#
# Recommended minimum Access Permission configuration:
#
# Deny requests to certain unsafe ports
http_access deny !Safe_ports

# Deny CONNECT to other than secure SSL ports
http_access deny CONNECT !SSL_ports

# Only allow cachemgr access from localhost
http_access allow localhost manager
http_access deny manager

# We strongly recommend the following be uncommented to protect innocent
# web applications running on the proxy server who think the only
# one who can access services on "localhost" is a local user
#http_access deny to_localhost
# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS
# Example rule allowing access from your local networks.
# Adapt localnet in the ACL section to list your (internal) IP networks
# from where browsing should be allowed
http_access allow localnet
http_access allow localhost
# And finally deny all other access to this proxy
http_access deny all
```



```
# Squid normally listens to port 3128
http_port 3128
```

```
# Uncomment and adjust the following to add a disk cache directory.
#cache_dir ufs /var/spool/squid 100 16 256
# Leave coredumps in the first cache dir
coredump_dir /var/spool/squid
# Add any of your own refresh_pattern entries above these.
refresh_pattern ^ftp:           1440    20%     10080
refresh_pattern ^gopher:        1440    0%      1440
refresh_pattern -i (/cgi-bin/|\?) 0     0%      0
refresh_pattern .               0       20%     4320
```

Appendix -3- Index.html file

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8"/>
   <script src="js/dash.all.debugmod.js"></script>
   <script>
      var player,firstLoad = true;
var info={
   timeOpened:new Date(),
   timezone:(new Date()).getTimezoneOffset()/60,
   pageon(){return window.location.pathname},
   referrer(){return document.referrer},
   previousSites(){return history.length},
   browserName(){return navigator.appName},
   browserEngine(){return navigator.product},
   browserVersion1a(){return navigator.appVersion},
   browserVersion1b(){return navigator.userAgent},
   browserLanguage(){return navigator.language},
   browserOnline(){return navigator.onLine},
   browserPlatform(){return navigator.platform},
   javaEnabled(){return navigator.javaEnabled()},
   dataCookiesEnabled(){return navigator.cookieEnabled},
   dataCookies1(){return document.cookie},
   dataCookies2(){return decodeURIComponent(document.cookie.split(";"))},
   dataStorage(){return localStorage},
   sizeScreenW(){return screen.width},
   sizeScreenH(){return screen.height},
   sizeDocW(){return document.width},
   sizeDocH(){return document.height},
```

```
sizeInW(){return innerWidth},
sizeInH(){return innerHeight},
sizeAvailW(){return screen.availWidth},
sizeAvailH(){return screen.availHeight},
scrColorDepth(){return screen.colorDepth},
scrPixelDepth(){return screen.pixelDepth},
latitude(){return position.coords.latitude},
longitude(){return position.coords.longitude},
accuracy(){return position.coords.accuracy},
altitude(){return position.coords.altitude},
altitudeAccuracy(){return position.coords.altitudeAccuracy},
heading(){return position.coords.heading}
speed(){return position.coords.speed},
timestamp(){return position.timestamp},
};
   function init() {
      player = dashjs.MediaPlayer().create();
      player.getDebug().setLogToBrowserConsole(false);
           load(this);
   }
   function showEvent(e)
   {
   }
   function log() {
           var type = "video";
           var metrics = player.getMetricsFor(type);
      var dashMetrics = player.getDashMetrics();
           var streamInfo = player.getActiveStream().getStreamInfo()
           var periodIdx = streamInfo.index;
      var repSwitch = dashMetrics.getCurrentRepresentationSwitch(metrics);
      var bufferLevel = dashMetrics.getCurrentBufferLevel(metrics);
```

```
        var maxIndex = dashMetrics.getMaxIndexForBufferType(type, periodIdx);
        var index = player.getQualityFor(type);
        var  throughPut = player.getThroughPut(type);
        var bitrate = repSwitch ?
Math.round(dashMetrics.getBandwidthForRepresentation(repSwitch.to, periodIdx) / 1000) : NaN;
        var droppedFPS = dashMetrics.getCurrentDroppedFrames(metrics) ?
dashMetrics.getCurrentDroppedFrames(metrics).droppedFrames : 0;
            var output = type+" "+bufferLevel +" "+bitrate+" "+throughPut+"
"+navigator.appCodeName+" "+info.browserName()+" "+info.sizeScreenW()+" "+info.sizeScreenH();

             writeStats(output);
    }
    function setListener(eventName)
    {
        player.on(dashjs.MediaPlayer.events[eventName],showEvent);
        var element = document.createElement("input");
        element.type = "button";
        element.id = eventName;
        element.value = "Remove " + eventName;
        element.onclick = function() {
            player.off(dashjs.MediaPlayer.events[eventName],showEvent);
            document.getElementById("eventHolder").removeChild(element);
        };
        document.getElementById("eventHolder").appendChild(element);
    }
    function load(button)
   {
        var url = "bbb_1080p2400b_dash.mpd";
        if (!firstLoad)
        {
            player.attachSource(url);
        }
```

```
        else
        {
           firstLoad = false;
           player.initialize(document.querySelector("video"), url, true);
                    setInterval(log,500)
        }
      }
function writeStats(stats){
var util="";
var xmlhttp;
if (window.XMLHttpRequest)
 {// code for IE7+, Firefox, Chrome, Opera, Safari
 xmlhttp=new XMLHttpRequest();
 }
else
 {// code for IE6, IE5
 xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
 }
 xmlhttp.onreadystatechange=function()
 {
 if (xmlhttp.readyState==4 && xmlhttp.status==200)
  {
     util = xmlhttp.responseText;
  }
 }
xmlhttp.open("GET","writestats.php?stats="+stats,true);
xmlhttp.send();
}
   </script>
   <style>
      video {
```

```
            width: 640px;
            height: 360px;
        }
    </style>
    <body onload="init()">
            <div>
            <video controls="true">
            </video>
        </div>
    </body>
</html>
```